VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Aplikace pro plánování cest

# Travel Planning Application

2017 Bc. Jaroslav Surala

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Jaroslav Surala**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Aplikace pro plánování cest
Travel Planning Application

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit webovou aplikaci pro plánování cest (převážně automobilem). Aplikace bude schopna na základě uživatelem zadaných dat o místech, které chce navštívit, navrhnout ideální časový harmonogram. Vstupní data mohou obsahovat různá omezení (čas, pevně dané pořadí, atd.). Pro hledání časového harmonogramu budou použity biologicky inspirované metody. Aplikace bude dále schopná získávat informace z API třetích stran, které bude společně s harmonogramem cesty adekvátně uživateli prezentovat.

Hlavní body zadání:
1. Přehled existujících řešení v dané oblasti.
2. Výběr vhodné biologicky inspirované metody.
3. Návrh a implementace aplikace.
4. Výkonnostní a UX experimenty.

Seznam doporučené odborné literatury:

[1] FRANCISCO BAPTISTA PEREIRA, Jorge Tavares (eds. Bio-inspired algorithms for the vehicle routing problem. Berlin: Springer, 2008. ISBN 9783540851516.
[2] DORIGO, Marco. Ant colony optimization. Cambridge: MIT Press, 2004, xiv, 305 s. ISBN 0-262-04219-3.
[3] FREEMAN, Adam. Pro AngularJS. New York: Apress, 2014. ISBN 9781430264484.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Janoušek**

Datum zadání: 01.09.2015
Datum odevzdání: 28.04.2017


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. dubna 2017 ..... Smala .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 20. dubna 2017 ..... Svrdla .....

Rád bych poděkoval panu Ing. Janu Janouškovi, za odborné vedení a rady při zpracovávání této diplomové práce.

**Abstrakt**

Cílem této diplomové práce je vytvořit aplikaci pro plánování cest (převážně automobilem), s pomocí nejmodernějších technologií pro tvorbu webových aplikací. Hlavním úkolem aplikace je navrhnout vhodný časový harmonogram na základě zadaných dat o místech, která mají být navštívena. Pro hledání budou použity biologicky inspirované metody.

První část práce se zaměřuje na porovnání existujících řešení pro optimalizaci cest. Dále se první část zabývá představením dvou biologicky inspirovaných metod a to Ant Colony Optimalization a Inteligent Water Drop Algorithm.

Poslední část práce se zabývá implementací aplikace, pro správu míst a zobrazení optimální trasy pro uživatelem zadaná místa. Nejprve se zaměřuje na implementaci správy míst a získáváním potřebných dat o jednotlivých místech. Dále se věnuje implementací biologicky inspirovaných metod pro optimalizaci trasy a následné vizualizaci optimální trasy na mapě, a také zobrazení časového harmonogramu cesty. Poslední část je věnována výkonnostnímu testování obou implementovaných metod.

**Klíčová slova**: Ant Colony Optimalizatin, Inteligent Water Drop Algorithm, AngularJS, ASP. NET Web API 2, Google Maps API

**Abstract**

The aim of this theses is to create travel planning aplication (mostly by car), based on modern technologies for web development. The main goal is to design suitable time schedule, based on information about places which are to be visited. For computation will be used biologically inspired methods.

The first part deals with comparison of existing solutions for travel planning. Also two biologically inspired methods are introduced in first part one is Ant Colony Optimalization and the second one is Water Drop Algorithm.

The final part delas with implementation of application for places management and view the optimal route for a user-specified places. First, it focuses on the implementation of places management and obtaining information about places. The last part also deals with implementation of both biologically inspired methods, and visualisation of optimal route on the map and also display the time schedule of route. The last part is dedicated for performance testing of implementations of both biologically inspired methods.

**Key Words**: Ant Colony Optimalizatin, Inteligent Water Drop Algorithm, AngularJS, ASP. NET Web API 2, Google Maps API

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| ACO | – | Ant Colony Optimization |
| TSP | – | Travelling Salesman Problem (Problém obchodního cestujícího) |
| IWDA | – | Intelligent Water Drops Algorithm |
| IWDs | – | Intelligent Water Drops |
| XML | – | Extensible Markup Language |
| JSON | – | JavaScript Object Notation |
| MVC | – | Model View Controller |
| HTML | – | HyperText Markup Language |
| CSS | – | Cascading Style Sheets |
| HTTP | – | Hypertext Transfer Protocol |
| SQL | – | Structured Query Language |
| ORM | – | Object relational mapping |
| DOM | – | Document Object Model |
| LINQ | – | Language Integrated Query |
| MSSQL | – | Microsoft SQL Server |
| API | – | Application Programming Interface |
| FTP | – | File Transfer Protocol |
| FPS | – | Frames Per Second |

# Seznam obrázků

# Seznam tabulek

# Seznam výpisů zdrojového kódu

# 1 Úvod

Od roku 1930, kdy australský ekonom Karl Menger definoval poprvé problém obchodního cestujícího, uběhlo již spousta času a událo se dost věcí. Postupně byla úloha řešena různými matematiky, se snahou zvládnout optimalizaci cesty pro co největší počet navštívených míst. Velký pokrok byl učiněn v letech 1970 a 1980, kdy se podařilo řešit přesně 2392 měst. V dnešní době jsme schopni řešit přesně problémy s 85 900 městy.

Jelikož se jedná o složitý problém, pro který doposud nebyl nalezen efektivní matematický algoritmus, začaly vznikat v uplynulých desetiletích aproximativní postupy. Jedny z nejmladších algoritmů zabývajícími se tímto problémem využívají principy, které se inspirují v evolučních procesech, jsou to tzv. biologicky inspirované algoritmy. Tyto moderní algoritmy jsou schopné řešit extrémně velké problémy (milion měst) v rozumném čase a s vysokou pravděpodobností, že výsledek nebude o mnoho odlišní od optimálního.

Díky tomu, že lidé více cestují automobily po světě, a vznikají firmy, které se specializují na přepravu, stává se řešení tohoto problému součástí každého dne. Začaly vznikat firmy, které se specializují na vyřešení optimalizace trasy, a aplikace, které pomáhají především ušetřit náklady na přepravu. Začaly vznikat mapové aplikace s možností plánování trasy s různými parametry, které ovlivní výslednou trasu. Tyto mapové aplikace jsou dostupné pro širokou veřejnost zdarma, lze je tak využívat i pro plánovaní soukromých cest.

Tato diplomová práce bude zaměřena na biologicky inspirované metody, které dokáží určit s velkou pravděpodobností optimální trasu zadanými místy. V první části představím webové aplikace, které řeší optimalizaci cesty mezi místy, a provedu porovnání těchto aplikací. Ve druhé části se pokusím přiblížit a detailně popsat fungování dvou biologicky inspirovaných algoritmů. Prvním z nich je Ant Colony Optimalizaton a druhým je Intelligent Water Drops Algorithm.

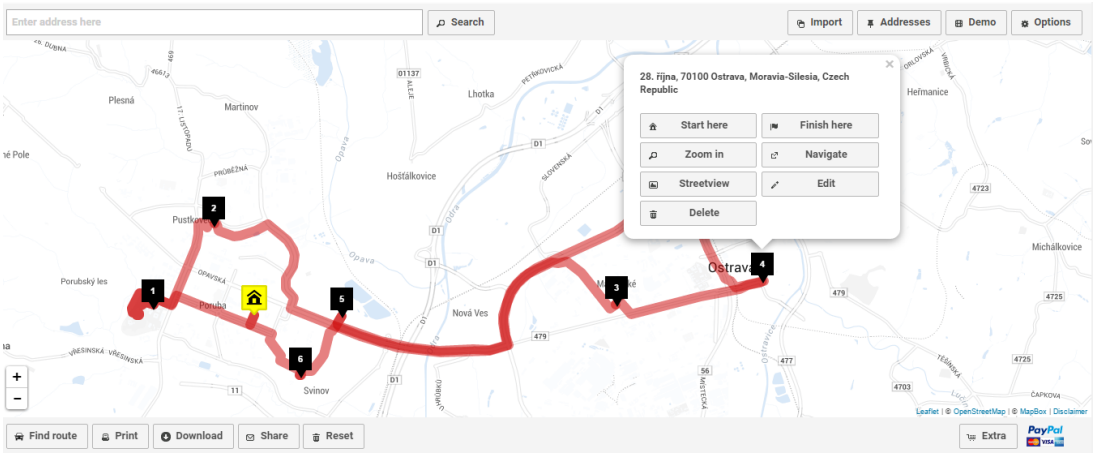
Poslední část práce je zaměřena na vývoj aplikace, pro plánování cest mezi místy. Pokusím se přiblížit, jak získavat potřebná data pro výpočet optimální trasy. Aplikace bude obsahovat implementaci obou biologicky inspirovaných algoritmů. Aplikace bude postavena na moderních technologiích jako je ASP. NET Web API 2, AngularJS, EntityFramework 6. Některé z využitých technologií budou i popsány.

# 2 Plánovače tras

V dnešní době je přeprava automobilem velice běžná. Ať už se jedná o přepravu do práce, či rodinný výlet, nebo jste řidičem rozvážkové služby. Pro firmy, které provozují rozvážkové služby, nebo zásobování pro různé obchodní řetězce či podniky je velice důležité najít co možná nejkratší cestu, aby snížili náklady na provoz vozidel. Pro tyto účely slouží plánovače tras, které dokaží najít nejkratší nebo nejrychlejší cestu mezi místy. Plánovače dokáží také upozornit na případné uzavírky nebo hustotu provozu na cestě. Plánovače dokáží také odhadnout spotřebu paliva, či kolik bude stát mýtné, nebo se zpoplatněným úsekům úplně vyhnout. Navíc existují i plánovače, které dokáží také pro sadu míst najít optimální cestu pro jejich průjezd tak, aby každé místo bylo navštíveno právě jednou za co nejkratší cestu. V následujícím textu si představíme některé online plánovače tras, které dokáží tuto optimální cestu nalézt.

## 2.1 RouteXL plánovač

Plánovač je dostupný na adrese https://www.routexl.nl/?lang=en. RouteXL dokáže navrhnout cestu všemi místy v té nejlepší možné variantě. Je to free plánovač pro cesty, avšak s omezením na maximálně dvacet adres na jednu cestu. RouteXL nabízí rozšíření toho omezení až na 150 míst na jednu cestu, ale je nutné si zaplatit buďto 10€ na jeden den, nebo 70€ na měsíc. Celý ceník lze nalést na webu https://www.routexl.nl/blog/pricing/?lang=en. Pro free verzi i placené verze platí, že počet cest, které chcete optimalizovat, je neomezený. RouteXL dokonce také poskytuje API pro programátory, které je blíže popsané na stránce https://www.routexl.nl/blog/api/?lang=en i se všemi omezeními.




Obrázek 1: Optimalizovaná trasa v RouteXL

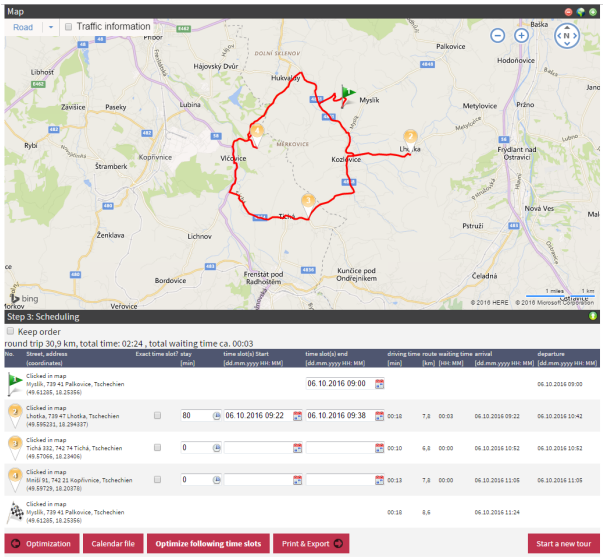
Práce s plánovačem je velice intuitivní, na zobrazené mapě lze rovnou klikem myši označit místo nebo adresu zadat do vyhledávacího pole. Adresy lze také naimportovat přes tlačítko v menu, kdy na každém řádku je jedna adresa. Plánovač má také různá nastavení, například jakým dopravním prostředkem cestujeme, jestli chceme cestu optimalizovat a nebo také v kolik hodin si

přejeme vyrazit. Dále také můžeme nastavit parametry, jako jsou jednotky, v kterých chceme, aby plánovač počítal, nebo kolik má náš dopravní prostředek spotřebu paliva na kilometr. Všechny tyto možnosti nastavení a jejich vysvětlení jsou dostupné v přehledné uživatelské příručce na adrese https://wiki.routexl.com/index.php/User manual. Na obrázku 1 je vidět, jak plánovač zobrazí optimalizovaný výsledek. Dále je zde vidět nabídka dostupná po kliknutí na vybrané místo. Například lze z této nabídky určit, že místo je počáteční pro trasu. Plánovač dokáže výslednou trasu vytisknout i s podrobnými detaily. Plánovač také dokáže stáhnout vyhledanou trasu do různých formátů pro automobilové navigace (dostupné formáty jsou INT, OV2 GPX, GPS, KML, CSV, TXT) a dále je schopný tuto trasu otevřít například v Google Maps.

## 2.2 MultiRoute plánovač

Plánovač je dostupný na adrese https://www.multiroute.de/?locale=en. Tento plánovač používá BING Maps API, opět se jedná o free plánovač, ale i zde s omezením. Ve free verzi je maximální počet míst sedm pro jednu cestu. Plánovač nabízí možnost rozšíření tohoto omezení až na 100 míst na jednu cestu, ale je nutné zaplatit. Plánovač nabízí 60minutové předplatné za 1,99€, nebo jednodenní za 4,99€, další možností je za 49€ měsíční.

Přidávání míst na mapu lze provádět kliknutím myši na mapu, nebo nahrát excelový soubor (.xsls, .xls, .ods), který obsahuje seznam adres. Formát takového souboru je popsán v sekci Help na webu plánovače. Adresy také můžeme zadat do textového pole pod mapou, kdy na každém řádku je jedna adresa. Plánovač může cestu přes místa zobrazit v takovém pořadí, které dostal, a nebo může trasu optimalizovat. Plánovač má dvě možnosti pro dopravní prostředek, a to automobil nebo pěšky. Dále si při optimalizaci můžeme vybrat zda chceme nejkratší cestu nebo nejrychlejší.




Obrázek 2: Optimalizovaná trasa v MultiRoute

Můžeme místu také nastavit, jak dlouho na daném místě strávíme, případně časové okno, kdy musíme na místě být. Plánovač tyhle údaje vezme v potaz a najde optimální cestu. Na obrázku 2 je vidět výsledná optimalizovaná trasa a část pod mapou, kde je možné nastavovat časové údaje jednotlivých míst. Trasu si můžeme vytisknout bohužel jen ve formátu, který je vidět na obrázku. Plánovač alespoň nabízí možnost stažení trasy pro GPS navigace v různých formátech. Dále plánovač poskytuje možnost exportovat naplánovanou cestu do souboru, který podporuje kalendář, jako je třeba v MS Outlook nebo Google Calendar.

## 2.3 Srovnání plánovačů

Na internetu nalezneme spouty plánovačů tras. Spousta z nich ovšem neumí ze zadané množiny míst vytvořit optimální cestu pro projetí. Dokáží pouze navrhnout cestu v zadaném pořadí. Vybrané plánovače tuto možnost mají. Plánovače nabízí podobnou funkčnost, ale jsou zde i nějaké rozdíly, které jsou zobrazeny v tabulce 1. Plánovače umožňují vytvoření účtu a díky tomu lze uložit již vyhledané a optimalizované trasy.

Tabulka 1: Srovnání plánovačů

| | RouteXL | MultiRoute |
|---|---|---|
| Free | Ano | Ano |
| Max počet míst Free | 20 | 7 |
| Max počet míst placená | 150 | 100 |
| Cena na měsíc (€) | 70 | 49 |
| Pole pro vyhledávání adres | Ano | Ne |
| Možnost importu adres | Ne | Ano |

MultiRoute plánovač nabízí možnost konkrétnímu místu přiřadit časový slot, kdy potřebujete v místě být, a podle toho optimalizuje cestu. RouteXL nabízí podrobnější detaily o naplánované cestě než plánovač MultiRoute. MultiRoute naopak podporuje vetší množství formátů, do kterých je schopný optimalizovanou trasu vyexportovat. O něco lépe si vede plánovač RouteXl, který ve free verzi podporuje optimalizaci cesty pro více míst. Pro běžného uživatele, který si chce naplánovat výlet se svou rodinou, je to dostačující počet.

# 3 Biologicky inspirované metody

Biologicky inspirované metody neboli evoluční metody patří mezi globálně optimalizační algoritmy. Cílem je prohledat prostor a najít optimální řešení, které maximalizuje nebo minimalizuje známou funkci. Pokud je prostor dostatečně malý, lze řešení nalézt i hrubou silou a vybrat to nejlepší. Ve většině případů bohužel velikost prostoru roste velmi rychle s velikostí problému. U takových problémů je tedy nemožné prohledávat celý prostor možných řešení, tedy je i nemožné najít optimální rešení v krátkém čase. Biologicky inspirované algoritmy tyto problémy dokážou řešit. [2]

Tyto algoritmy je vhodné použít na obtížné problémy, které mají velký prostor řešení. Dále se algoritmy hodí na problémy, které jsou obtížně formulovatelné. Například víme jaký by měl být výstup, ale nevíme jakým způsobem takový výstup dostat. Příkladem takového problému může být návrh elektronických schémat. Víme, jaký požadujeme od schématu výstup, ale nevíme jak správně propojit součástky, abychom tento výstup dostali. U obtížných problémů ovšem tyto algoritmy jsou schopny najít pouze přibližná řešení. U těchto problémů nemusí výpočet být rychlý. [2]

## 3.1 Problém obchodního cestujícího

Jeden z problémů, který dokáži biologicky inspirované algortimy vyřešit v reálném čase s dobrými výsledky, je Traveling Salesman Problem. TSP je optimalizační problém, kdy chceme ze startovního místa najít nejkratší cestu, skrz všechna místa a zároveň je navštívit pouze jednou a vrátit se do místa, ze kterého započala cesta. TSP problém lze matematicky vyjádřit jako úplný neorientovaný graf G = (V, E) s ohodnocením hran, kde V je množina vrcholů a E je množina hran, které spojují vrcholy. Každá hrana $(i, j) \in E$ má přiřazenu váhu $d_{ij}$, která reprezentuje vzdálenost mezi místy $j$ a $i$. TSP problém spočívá v nalezení nejkratší Hamiltonovské kružnice v grafu, kde Hamiltonovská kružnice je uzavřená cesta, která projde každý vrchol právě jednou v grafu G. [3]

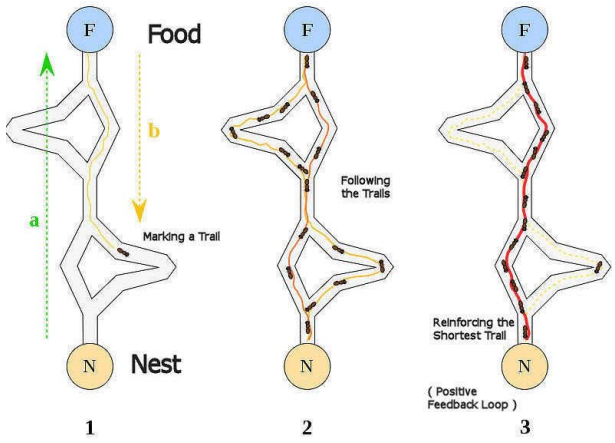
## 3.2 Ant Colony Optimization

Ant Colony Optimization byl poprvé představen Marcem Dorigem v jeho disertační práci v roce 1992. ACO je metaheuristika, shrnující poznatky ze studia různých druhů mravenců. ACO popisuje jeden dobrý přístup k řešení výpočetně, časově náročných problémů. Používá množství jednoduchých výpočetních agentů, kteří spolupracují při hledání globálního optimálního řešení. Časová složitost ACO algoritmů je běžně kvadratická, může být i lineární. ACO může být použito pro řešení jakéhokoli diskrétního kombinačního problému, pro který lze použít nějakou konstruktivní heuristickou proceduru. Aby bylo možné ACO použít, je nutné problém převést na prohledávání grafu. Algoritmus je inspirovaný skutečnými mravenci v koloniích při hledání a sbírání potravy. [3]

### 3.2.1 Chování mravenců

Mravenci se v přírodě pohybují náhodně, když naleznou potravu vrací se do mraveníště, a zanechávají za sebou feromon. Jakmile další mravenec narazí na feromonovou stopu, tak přestane chodit náhodně a začne se držet feromonové stopy. Čím více feromonu na cestě, tím roste pravděpodobnost, že si mravenec vybere feromonovou cestu. V okamžiku, kdy další mravenec nalezne potravu, začne také vypouštět feromon. Cesta je intenzivněji značkována a přitahuje další mravence, protože jejich úkolem je donést do mraveniště potravu za co nejkratší čas tedy nejkratší cestu. Kratší cesta, kterou mravenci absolvují vícekrát, má silnější stopu feromonu než cesta, která do mraveniště trvá déle. Na delší trase feromon postupně vyprchává a cesta pro ostatní mravence přestává být atraktivní. Vyprchávání feromonu je velmi důležité, protože jinak by hledání konvergovalo předčasně jen do jedné cesty, často do lokálního extrému. [3, 4]

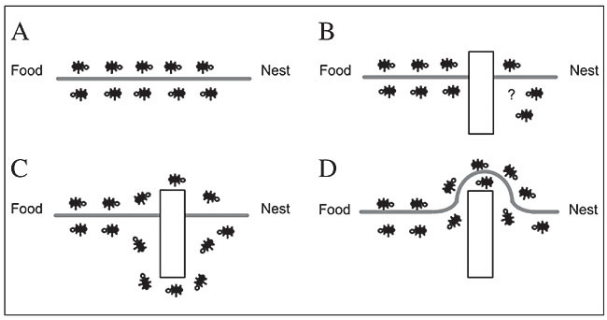
Popis tohoto chování vychází z laboratorního pokusu známého jako experiment s dvojitým mostem (Double Bridge Experiment). Během pokusu s dvojitým mostem bylo hnízdo mravenců druhu *Limepithemahumile* propojeno se zdrojem potravy a dvěma trasami. Mravenci se časem naučili používat kratší trasu a když jim byla nabídnuta nová ještě kratší trasa nedokázali ji využít. Mravenčí kolonie uvázla v lokálním extrému řešeného problému. Feromonová stopa na nejkratší cestě byla tak výrazná, že se novou cestou vydávalo příliš málo mravenců, aby ji označili jako nejvýhodnější. Další experimenty ukázaly, že různé druhy mravenců používají různé metody pro volbu trasy. Obrázek č. 1 znázorňuje chování mravenců v přírodě. [4]




Obrázek 3: Pohyb mravenců v přírodě (zdroj: https://cs.wikipedia.org/wiki/Mravencovití)

Pokud na cestě nejsou překážky, chodí mravenci rovně pro jídlo a zpět. V přírodě se ovšem vyskytuji překážky, takže mravenci jsou nuceni je obcházet. V takovém případě mravenci překážku obchází oběma směry. Cesty nebyly prozkoumány a nebyl roznesen feromon. Po delší době ovšem na kratší cestě bude více feromonu a na delší cestě se začne vypařovat. [5]

Na obrázku 4 jsou uvedeny různé situace cest mravenců a feromonové stopy. Cesta A naznačuje feromon, který nemá mezi mraveništěm žádnou překážku. Situace B, kdy je mezi potravou a mraveništěm překážka, naznačuje přerušení feromonu. Situace C ukazuje, že mravenci našli




Obrázek 4: Mravenci a překážky (zdroj: http://www.funpecrp.com.br/gmr/year2005/vol3-4/wob09_full_text.htm)

dvě cesty okolo překážky. Situace D znázorňuje nalezení a "zvýhodnění"kratší cesty feromonem. [5]

### 3.2.2 Kroky ACO

Po počáteční inicializaci následují tři fáze. V každé iteraci všichni mravenci současně a asynchronně projdou graf problému. Poté se nechá vyprchat část feromonů a na prošlé cesty se přídá nová fermonová stopa. Následovat mohou centrálně řízené akce, které nemůže provést pouze jediný mravenec. Příkladem může být použití metod lokálního vyhledávání v nalezeném řešení. Kostra ACO algoritmů je na výpise 1. [4, 6]

```
Initialize
While Stopping critera is not satisfied do
      AntBasedSolutionConstruction()
      PheromoneUpdate()
      DaemonActions() {optional}
End While
```

Výpis 1: Pseudo kód hledání řešení ACO algoritmy

**AntBasedSolutionConstruction:** Množinu mravenců lze považovat za konstruktivní pravděpodobnostní heuristiku, která sestaví řešení jako sekvenci z hodnot komponent řešení. Konstrukce řešení započne z prázdného částečného řešení a je postupně rozšiřováno o další komponenty, které splňují podmínky omezení. Konstruce řešení znamená průchod grafem. Výběr komponenty z množiny komponent je stochastická metoda, která je závislá na feromonu.[4]
**PheromoneUpdate:** Odlišné varianty ACO se liší především tím, jak aktualizují hodnotu feromonu. Cílem aktualizace feromonu je zvýšit hodnotu pro dobrá řešení a naopak pro špatná řešení snížit hodnotu. Toho dosáhneme vypařováním feromonu. [4]

### 3.2.3 Ant System

Ant System je jeden z prvních ACO algoritmů. Algoritmus napodobuje vlastnosti a způsob chování mravenců se simulovanými mravenci, kteří se pohybují po grafu reprezentující problém. Na začátku algoritmu se nejprve inicializují hodnoty feromonů, které jsou nastaveny tak, aby byly mírně vyšší než očekávané hodnoty položené v prvních běžích algoritmu. Mravenci se pohybují mezi jednotlivými uzly s pravděpodobností výběru dalšího místa podle rovnice uvedené níže: [3]

$$p_{i,j} = \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum\limits_{l \in N_i^k} [\tau_{i,l}]^{\alpha}[\eta_{i,l}]^{\beta}} \tag{1}$$

Kde $j \in N_i^k$, $N_i^k$ je množina přípustných uzlů (neobsahuje již navštívené), $\tau_{i,j}$ je množství feromonu na hraně $i, j$ mezi uzly $\alpha$ je parametr kontroly vlivu na $\tau_{i,j}$. $\eta_{i,j}$ je požadavek na hranu $i, j$ (typicky $1/d_{i,j}$), $\beta$ je parametr kontroly vlivu na $\eta_{i,j}$. Parametr $\alpha$ zvětšuje rozdíly v hodnotách feromonů, velké $\beta$ posílí význam feromonů. Malé $\alpha$ podpoří průzkum grafu. Pokud snížíme parametr $\beta$, svádí to mravence do bližších měst. [3]

Po dokončení cest všemi mravenci se nechá část feromonů vypařit. Přepočet vypařování feromonu je proveden přes následující vztah:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} \tag{2}$$

kde $\rho$ je poměr vypařování feromonu. Dalším krokem je položení nových feromonů podle rovnice

$$\tau_{i,j} = \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} \tag{3}$$

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/C_k & \text{pokud hrana } (i,j) \in T^k \\ 0 & \text{jinak} \end{cases}$$

kde $\tau_{i,j}$ je množsví feromonu na hraně $i, j$ mezi uzly, $m$ je počet mravenců, $T^k$ je cesta mravence $k$ a $C^k$ je délka cesty. Nastavení parametrů je důležité. Experimentálně bylo zjištěno, že vhodná hodnota pro $\alpha$ je 1, $\beta$ mezi 2 a 5 a pro parametr $\rho$ 0.5.

```
Generate initial pheromone matrix P
iteration = 0
While Stopping critera is not satisfied do
      Deploy m ants randomly on the graph nodes
      Repeat
         For each ant do
            Go ahead n steps in graph. Choose next node by eq 1
            Apply step by step pheromone update in matrix P
         End for
      Until every ant has build solution
      Update best solution
      Scald pheromones in matrix P by eq 3
      iteration = iteration + 1
```

Výpis 2: Pseudo kód algoritmu Ant System

Mravenčí optimalizace probíhá v několika po sobě jdoucích iteracích. V každé iteraci provede $m$ mravenčích agentů své úkoly a vyhodnotí se cesty, které našli. Algoritmus končí splněním ukončujících podmínek, které mohou být nalezení optimálního řešení, určitý počet iterací bez zlepšení nebo dosažení určitého množství iterací. Tento základní mravenčí algoritmus se postupem času dočkal velkého množství modifikací. [3]

### 3.2.4 Ellitist Ant System

Je první vylepšení AS. Algoritmus přináší oproti algoritmu Ant System změnu v pokládání feromonů. Algoritmus dodatečně položí feromony na hrany, jež jsou součástí do té doby nejlepší známé nalezené cesty. Toto chování zpravidla urychluje konvergenci algoritmu, ale zvyšuje pravděpodobnost uváznutí v lokálním extrému řešeného problému. Modifikovaná rovnice pro úpravu feromonové matice vypadá následovně: [7] [3]

$$\tau_{i,j} = \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} + e\Delta\tau_{i,j}^{bs} \tag{4}$$

$$\Delta\tau_{i,j}^{bs} = \begin{cases} 1/C_{bs} & \text{pokud hrana } (i,j) \in T^{bs} \\ 0 & \text{jinak} \end{cases} \tag{5}$$

kde $\Delta\tau_{i,j}^{k}$ představuje přírustek feromonu na hraně $a_{ij}$ uložený $k$ - tým mravencem vyjádřený podle rovnice 3, $C^{bs}$ je délka nejlepší cesty a $e$ je parametr. Výsledky experimentů ukazují, že tato změna společně s vhodně zvoleným $e$ vede k nalezení lepších řešení v kratším čase. [7][3]

### 3.2.5 $MAX - MIN$ Ant System

$MM$AS mění Ant Systém ve 4 hlavních bodech.

- Velký důraz je kladen na nejlepší nalezenou cestu. Pouze nejlepší mravenec v dané iteraci (hlavního cyklu algoritmu) nebo do té doby nejlepší mravenec položí feromony. [8][3]
- Tato strategie by mohla vést ke stagnaci. Celá kolonie by se mohla pohybovat pouze po jediné sice dobré ale suboptimální cestě vlivem velkého množství feromonů hromaděného na jedné hraně trase. $MM$AS tento problém řeší zavedením horního ($\tau_{max}$) a dolního ($\tau_{min}$) limitu na množství feromonu na jedné hraně grafu. [8][3]
- Feromonové stopy jsou na začátku inicializovány na hodnotu horního limitu přípustného na hraně grafu. Toto společně s pomalejším vyprcháváním vede k intenzivnějšímu průzkumu v začátcích běhu algoritmu. [8][3]
- Pokud systém projeví stagnaci nebo nebylo nalezeno nové řešení během daného počtu iterací, jsou feromonové stopy reinicializovány horního limitu přípustného feromonu na hraně. [8][3]

### 3.2.6 Ant Colony System

ACS rozšiřuje AS ve 3 hlavních bodech.

- Mravenci používají agresivnější pravidlo pro výběr příštího uzlu v grafu. S pravděpodobností $q_0$ (parametr) sledují nejlepší známou cestu z uzlu $i$. S opačnou pravděpodobností použijí pravidlo z AS. Volbou $q_0$ lze podporovat průzkum grafu nebo koncentrovat hledání řešení okolo nejlepší známé cesty. [9]
- Pouze nejlepší mravenec pokládá svou feromonovou stopu a pouze jeho stopa vyprchává pomocí rovnice 6. Tato změna snižuje složitost aktualizace feromonových stop z kvadratické na lineární. [9]
- Každý mravenec bezprostředně po každém svém kroku z uzlu $i$ do uzlu $j$ aktualizuje množství feromonu na hraně $(i, j)$ rovnicí 7, kde $\xi \in (0, 1)$ experimentálně zjištěná dobrá hodnota je 0.1, $\tau_0$ je inicializační hodnota feromonů. Díky takové aktualizaci dojde ke snížení množství feromonů, což podpoří volbu ostatních uzlů a zabrání stagnaci. [9]

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \rho\Delta\tau_{i,j}^{bs} \tag{6}$$

$$\tau_{i,j} = (1 - \xi)\tau_{i,j} + \xi\tau_0 \tag{7}$$

Existují další modifikace a varianty mravenčích algoritmů. Jsou úspěšné ve vyhledávání téměř optimálních cest v komplexních grafech. Výborných výsledků dosahují mravenčí algo-

ritmy zejména, pokud je pro řešení konkrétního problému k dispozici vhodná a-priori heuristická informace, která může být během výpočtu využita.

Mravenčí algoritmy jsou jednou z nejúspěšnějších bio-inspirovaných metaheuristik. Tento úspěch podpořil další studium biologických systémů a návrh podobných algoritmů napodobujících chování živočichů v přirodě, např. včel [10] nebo světlušek [11].

### 3.2.7 Aplikace mravenčích algoritmů

Optimalizace pomocí mravenčích algoritmů byla použita pro celou řadu praktických úloh. Mezi prvními, kdo využil algoritmus založený na mravenčí heuristice, byla firma EuroBois. Ta aplikovala algoritmus na řadu plánovacích úloh, jako je třeba nepřetržitý two-stage flow shop s konečným počtem nádrží. Modelovaný problém zahrnoval řadu reálných omezení jako jsou časy nastavení, kapacita nádrží, kompatibilita zdrojů a kalendář údržby [3]. Další společnost, která hraje velmi důležitou roli v propagaci reálného využití mravenčích algoritmů, je AntOptima. Společnost vyvinula sadu nástrojů pro řešení vehicle routing problémů, které jsou založeny na mravenčích algoritmech. Zejména úspěšný je produkt DyvOil, který je určen pro řízení a optimalizaci distribuce topného oleje. Další zajímavý produkt firmy je AntRoute, který je určen pro optimalizaci velkého počtu tras vozidel a flotil [3]. K plánování údržby flotil dopravních prostředků využili mravenčí algoritmy Abrahão a Gualda. Problém plánování údržby spočívá v nalezení optimální sekvence preventivních prohlídek tak, aby byla maximalizována životnost a využití vozidel [12].

## 3.3 Intelligent Water Drops Algorithm

Algoritmus byl poprvé představen Shahem-Hosseini v roce 2007. Algoritmus je založen na dynamice říčních systémů (hledáním trasy od pramene k ústí) a činností, které se dějí mezi vodními kapkami (IWD). Hlavními vlastnostmi jednotlivých IWD jsou rychlost (velocity) a půda (soil). [13] [1]

Obě vlastnosti se mění v průběhu životního cyklu IWD. IWD se pohybuje od zdroje k cíli. Velocity a soil IWD jsou zpočátku nula. Během své cesty se pohybuje v prostředí, z něhož odstraňuje nějakou půdu a může získat nějakou rychlost. Ze své současné pozice k další se zvyšuje rychlost IWD o částku nelineárně přímo úměrnou převrácené hodnotě *soil* mezi dvěma pozicemi. Proto cesta, na které je menší množství půdy, učiní IWD rychlejší než cesta, kde je více půdy. [13] [1]

Každá IWD během své cesty v prostředí shromažďuje půdu. Půda je odebírana z cesty, která spojuje dvě místa. Množství půdy, které je přidáno do IWD, je rovno nelinárně úměrné hodnotě převrácené hodnoty času, který IWD potřebovala na cestu mezi místy. Časový interval je spočítán pomocí základních zákonů fyziky pro lineární pohyb. To znamená, že čas je úměrný rychlosti IWD a nepřímo úměrný vzdálenosti mezi místy. Části prostředí, které používá více IWD, bude mít méně půdy. [13] [1]

Algoritmus obsahuje mechanismus výběru cesty do dalšího místa nebo kroku. Mechanismus preferuje cesty, které mají nízké množství půdy, spíše než ty, které mají vysoké množství půdy. Chování výběru cesty je implementováno zavedením náhodného rovnoměrného rozdělení veličiny půdy na dostupných cestách. Poté pravděpodobnost toho, že další krok bude vybraný, je nepřímo úměrný množství půdy na dostupných cestách. Proto cesty s menším množstvím půdy mají větší šanci, že budou vybrány. [13] [1]

### 3.3.1 Parametry algoritmu

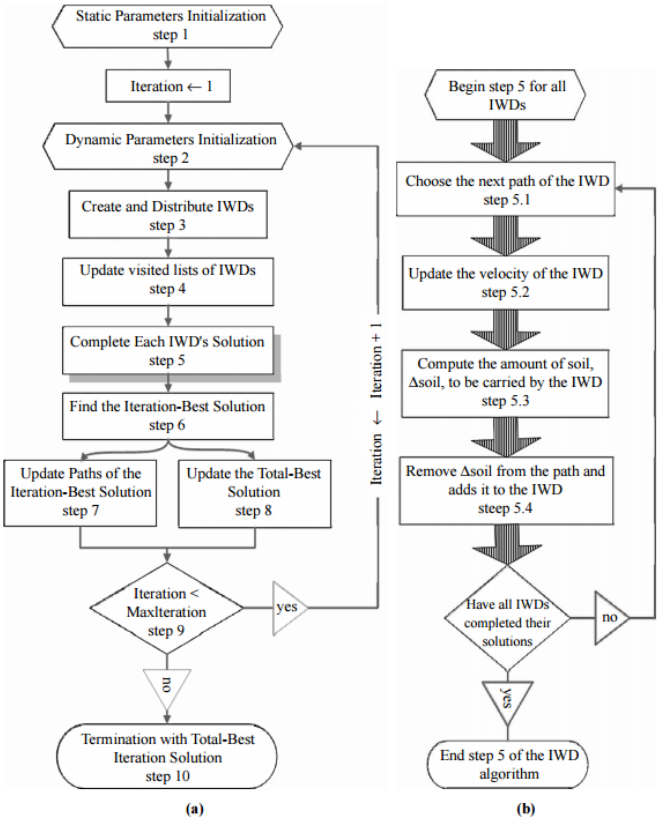
Algoritmus má dva druhy parametrů. Jeden druh jsou statické parametry, které se nezmění po celou dobu běhu algroitmu. Další druh jsou dynamické parametry, které se reinicializují pokaždé iteraci. Algoritmus obsahuje řadu statických parametrů. Parametr maximální počet iterací $iter_{max}$ je specifikovaný uživatelem. Počet water drops $N_{IWD}$ je nastaven na kladné celé číslo, většinou je nastaven na počet vrcholů v grafu (počet měst). Pro upravování vlastnosti *velocity* slouží parametry $a_v, b_v$ a $c_v$. Pro upravování vlastnosti *soil* slouží parametry $a_s, b_s$ a $c_s$. Pro aktualizaci hodnoty *soil* na hraně je použit parametr $p_n$. Je to malé kladné číslo v rozmezí 0 až 1. Pro aktualizaci vlastnosti *soil* na všech hranách slouží parametr $p_{IWD}$, který je také malé kladné číslo v rozmezí 0 až 1. Parametr $InitSoil$ je počáteční hodnota vlastnosti *soil* na každé hraně. Parametr $InitVel$ je počáteční rychlost každé IWD. [13] [1]

### 3.3.2 Konstrukce řešení

Algoritmus reprezentuje TSP problém ve formě grafu (N,E). N představuje sadu vrcholů, odpovídající městům. E představuje sadu hran, odpovídající cestám mezi jednotlivými městy. Každá IWD začíná konstrukcí svého řešení postupně cestováním mezi jednotlivými vrcholy, dokud nenavštíví všechny vrcholy. Jedna iterace algoritmu je dokončena, až všechny IWD dokončí své řešení. [13] [1]

Na obrázku vývojového diagramu můžeme vidět dvě části. První část (a) zachycuje hlavní kroky algoritmu. Část (b) je diagram kroků kroku 5 z hlavní části IWD algoritmu.

Po inicializaci všech statických parametrů začne algoritmus hledat optimální řešení problému. Algoritmus pracuje v iteracích, kdy na začátku se reinicializují dynamické parametry každé IWD. Každá IWD má na začátku každé iterace hodnotu velocity nastavenu na hodnotu parametru $InitVel$ a hodnotu soil nastavenu na 0. Každá IWD má seznam všech navštívených vrcholů (měst), které jsou před začátkem každé iterace nastaveny na prázdný seznam. [1] [13]




Obrázek 5: Diagram IWD algoritmu (zdroj: [1])

Nejprve rozložíme všechny IWD náhodně po grafu a nastavíme příslušný startovací vrchol jako navštívený. Každá IWD ve vrchlou $i$, si vybere další vrchol $j$, který neporušuje žadné pravidlo problému a není v seznamu navštívených míst $V_c(IWD)$ s pravděpodobností podle níže uvedené rovnice. [1] [13]

$$P_i^{IWD}(j) = \frac{f(soil(i,j))}{\sum_{k \notin V_c(IWD)} f(soil(i,k))} \tag{8}$$

$$f(soil(i,j)) = \frac{1}{\varepsilon_s + g(soil(i,j))} \tag{9}$$

Kde $\varepsilon_s$ je kladné číslo, aby se předešlo možným dělením nulou. V práci používám $\varepsilon_s$ rovno 0.0001.

$$g(soil(i,j)) = \begin{cases} \text{soil(i,j)} & pokud \min_{l \notin V_c(IWD)}(soil(i,l)) \geq 0 \\ \text{soil(i,j) -} \min_{l \notin V_c(IWD)}(soil(i,l)) & jinak \end{cases} \tag{10}$$

Poté je vrchol uložen do seznamu navštívených vrcholů a dále je aktualizována rychlost IWD podle rovnice.

$$vel^{IWD}(t+1) = vel^{IWD}(t) + \frac{a_v}{b_v + c_v \cdot soil^2(i,j)} \tag{11}$$

Kde $vel^{IWD}(t+1)$ je aktualizovaná rychlost IWD. Pro IWD pohybující se po cestě z vrcholu $i$ do vrcholu $j$, je spočítaná hodnota soil $\Delta soil(i,j)$, které nabere z cesty pomocí rovnice.

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s \cdot time^2(i,j;vel^{IWD}(t+1))} \tag{12}$$

Kde $time(.)$ je definovaný pomocí rovnice.

$$time(i,j;vel^{IWD}(t+1)) = \frac{HUD(j)}{vel^{IWD}(t+1)} \tag{13}$$

Kde heuristická nevhodnost $HUD(j)$ je vhodně definovaná pro daný problém. Pro problém TSP je definovaná jako podíl $distance(i,j)$ a $vel^{IWD}$. Po aktualizaci rychlosti IWD je nutné ještě aktualizovat hodnotu soil na cestě z vrcholu $i$ do vrcholu $j$ pomocí rovnice 14 a také vlastnost $soil^{lWD}$, kterou IWD nese pomocí rovnice 15 .

$$soil(i,j) = (1 - p_n) \cdot soil(i,j) - p_n \cdot \Delta soil(i,j) \tag{14}$$

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \tag{15}$$

Jakmile všechny IWD dokončí své řešení, je pomocí následující rovnice určeno nejlepší řešení dané iterace.

$$T^{IB} = arg\max q_{\forall T^{IWD}}(T^{IWD}) \tag{16}$$

Kde $q(.)$ je funkce, která meří kvalitu řešení. Pro problém TSP je za nejkvalitnější řešení označeno to, které urazí nejmenší vzdálenost při průchodu grafem.

Po nalezení nejkvalitnějšího řešení iterace je potřeba aktualizovat hodnotu *soil* na každé hraně dané cesty rovnicí.

$$soil(i,j) = (1 + p_{IWD}) \cdot soil(i,j) - p_{IWD} \cdot \frac{1}{N_{IB} - 1} \cdot soil_{IB}^{IWD} \quad \forall (i,j) \in T^{IB} \tag{17}$$

Kde $N_{IB}$ je počet vrcholů v řešení $T^{IB}$. Když máme určeno nejkvalitnější řešení iterace a aktualizovanou hodnotu *soil* na dané cestě, musíme aktualizovat dosud nejkvalitnější řešení rovnicí.

$$T^{TB} = \begin{cases} T^{TB} & q(T^{TB}) \geq q(T^{IB}) \\ T^{IB} & jinak \end{cases} \tag{18}$$

Následně zvýšíme číslo iterace o jedna a pokud je číslo iterace stále menší, opakujeme předešlé kroky. Algoritmus končí a vrací nejkvalitnější řešení $T^{TB}$ ve chvíli, kdy je číslo iterace vetší než parametr $iter_{max}$, který je definovaný. Nutno podoknout, že algoritmus najde optimální řešení, pokud je počet iterací dostatečne veliký. [13] [1] [14]

### 3.3.3 Experimentální výsledky

Pro měření výsledků algoritmu byly použity dobře známé TSP problémy (Berlin52, Eil51, Eil76, Eil101, KroA100, KroC100, Pr76, Lin105, St70, Ulysses22), které jsou pravidelně uváděny v literatuře. Pro testování budeme používat následující nastavení parametrů. [13]

Tabulka 2: Nastavení parametrů IWD algoritmu

| Parametr | Hodnota |
|---|---|
| $a_v$ | 1 |
| $b_v$ | 0.01 |
| $c_v$ | 1 |
| $a_s$ | 1 |
| $b_s$ | 0.01 |
| $c_s$ | 1 |
| $iter_{max}$ | 1000 |
| $InitSoil$ | 10000 |
| $InitVel$ | 200 |
| $p_{IWD}$ | 0.9 |
| $p_n$ | 0.9 |
| $N_{IWD}$ | počet měst |

Algoritmus byl spuštěn 50krát pro každý TSP problém s cílem ukázat konzistenci výsledků. Pro měření vzdáleností mezi městy je použita klasická euklidovská rovnice. Srovnání výsledků algoritmu s optimálními pro TSP problémy je zobrazeno v tabulce 3, tyto výsledky jsou převzaty z článku [13].

Hodnota sloupce Chyba, která je v tabulce 3, je spočítaná pomocí rovnice 19. Kde $X$ je délka, kterou našel algoritmus IWD, a $Y$ je optimální délka trasy.

$$Chyba = \frac{X - Y}{Y} * 100 \tag{19}$$

Nejmenší chybovost v řešených problémech byla zjištěna u problémů Berlin52, Eil51 and Ulysses s 0% chybovostí. Největší chybovost pak byla zjištěna u problému Eil101. Tabulka 3 ukazuje, že cesty získané algoritmem jsou stejné nebo velice blízké optimální cestě. Pro získání přehledu o výkonnosti algoritmu byl porovnán s ant colony optimalization. Počet mravenčích agentů byl nastaven na počet měst problému a ukončující kritérium bylo nastaveno na 1000 iterací. Algoritmus byl spuštěn 50krát na každý TSP problém stejně jako IWD. Výsledky jsou zobrazeny v tabulce 4. [13]

Tabulka 3: Srovnání mezi optimální délkou a IWD vypočítanou

| Problém | Počet měst | IWD | Optimální délka | Chyba (%) |
|---|---|---|---|---|
| Berlin52 | 52 | 7542 | 7542 | 0 |
| Eil51 | 51 | 426 | 426 | 0 |
| Eil76 | 76 | 540 | 538 | 0,37 |
| Eil101 | 101 | 639 | 629 | 1,59 |
| KroA100 | 100 | 21429 | 21282 | 0,69 |
| KroC100 | 100 | 20816 | 20749 | 0,32 |
| Lin105 | 105 | 14393 | 14379 | 0,10 |
| Pr76 | 76 | 109608 | 108159 | 1,34 |
| St70 | 70 | 676 | 675 | 0,15 |
| Ulysses22 | 22 | 72 | 72 | 0 |

Tabulka 4: Srovnání mezi ACO a IWD

| Problém | Počet měst | IWD | ACO |
|---|---|---|---|
| Berlin52 | 52 | **7542** | 7549 |
| Eil51 | 51 | **426** | 438 |
| Eil76 | 76 | **540** | 553 |
| Eil101 | 101 | **639** | 674 |
| KroA100 | 100 | **21429** | 22413 |
| KroC100 | 100 | **20816** | 21031 |
| Lin105 | 105 | **14393** | 14649 |
| Pr76 | 76 | **109608** | 113774 |
| St70 | 70 | **676** | 696 |
| Ulysses22 | 22 | **72** | 75 |

Z tabulky 4, která je převazta z článku [13], je patrné, že IWD algoritmus vypočítá lepší optimální cestu než ACO algoritmus pro všechny testované problémy. IWD algoritmus je optimalizační algoritmus, který používá roj vodních kapek pro nalezení nejlepšího řešení v prostředí řešeného problému. IWD algoritmus dokáže řešit problémy jako jsou n-queens puzzle, TSP a Multiple Knapsack problem. [13]

# 4 Použité technologie

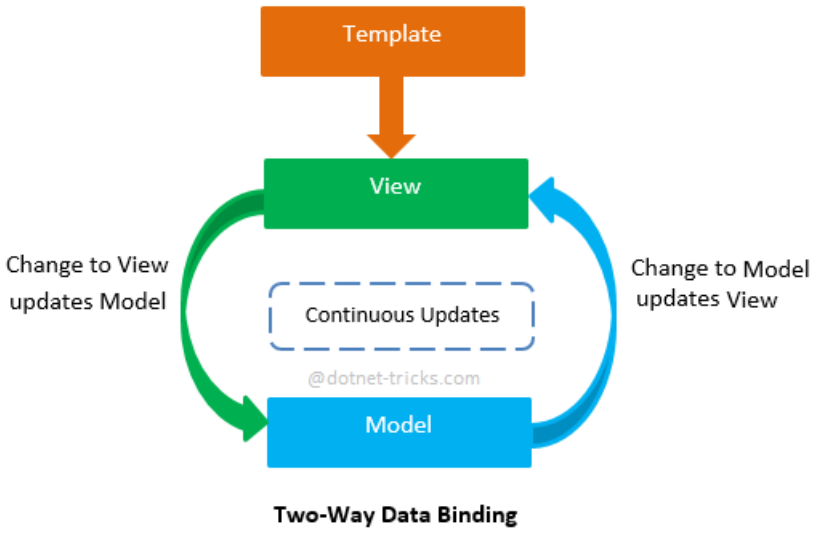
Aplikace se skládá z klientské části a serverové části. Pro klientskou část byl použit programovací jazyk JavaScript spolu s frameworkem AngularJS. Na serverovou část byl použit framework ASP.NET Web API 2 s programovacím jazykem C#. Data z aplikace se ukládají do MSSQL 2014 databáze a pro přístup k datům byl použit Entity Framwork 6. Celý vývoj probíhal ve Visual Studiu 2015 a zdrojové kódy byly verzovány za pomocí Team Foundation Version Control.

## 4.1 AngularJS

AngularJS je open source front-end web application framework kompletně napsán v JavaScriptu. Framework je udržován hlavně firmou Google, ale také komunitou jednotlivců. Framework obsahuje řadu zajímavých myšlenek, přičemž mezi nejúspěšnější se řadí Two Way Data-Binding, implementace Dependency Injection, testovatelnost, direktivy a znovu použitelnost komponent. AngularJS implementuje architekturu MVC. [15]

### 4.1.1 Two Way Data-Binding

Nejužitečnější vlastností frameworku je koncept Two Way Data-Binding, který řeší synchronizaci stavů mezi modelem a view. Jedná se o vlastnost, která například, když uživatel vyplní řetězec do textového pole, tento text přenese do modelu (první směr) a následně se z modelu propíše do ostatních částí view (druhý směr).




Obrázek 6: Two Way Data-Binding (zdroj: http://www.angularjstutorialsng.com/2016/07/angularjs-data-binding-ng-model.html)

```
<p> Enter name: <input type="text" ng-model="name"><br>
Hello <span ng-bind="name"></span>!
Hello <span>{{name}}!</span></p>
```

Výpis 3: Ukázka Two Way Data-Binding

Na výpise 3, je vidět jak AngularuJS provádímé, Two Way Data-Binding. Textovému poli přidáme direktivu **ng-model**, která říká, že cokoli napíšeme do textového pole se, přesune do modelu **name**. Druhý řádek obsahuje prvek span s direktivkou **ng-bind**, která říká, že do prvku span má vypsat data z modelu **name**. Nebo můžeme použít variantu, která je na třetím řádku, jedná se o Angular expression, který vyhodnotí model **name** a zobrazí výsledek.

### 4.1.2 Dependency Injection

AngularJS obsahuje zabudovaný Dependency Injection mechanismus, který řeší resolvování závislostí napříč celou vyvíjenou aplikací. Umožňuje specifikovat další komponenty, které jsou potřeba v konkrétní komponentě.

```
app.controller('AllProjectsController', ['$scope', '$http', '$rootScope',
    function ($scope, $http, $rootScope) {
  // Code
}
```

Výpis 4: Ukázka deklarace controlleru

Controlleru na výpise 4 se předávají 3 parametry. Uvnitř controlleru se nepracuje s jinými závislostmi, než které jsou předány, a díky tomu se části aplikace dobře testují. Subsystém, který obsahuje AngularJS, jakmile vytváří novou instanci controlleru, zkontroluje všechny závislosti v deklaraci a předá je jako parametry.

Tyto závislosti se jmenují **services** (služby). Služby jsou dvojího typu buď interní, dodávané spolu s frameworkem, nebo si můžeme napsat vlastní služby. Interní služby jsou prefixovány znakem dolar ($). V uvedeném přikladu jsou použity 3 služby.

1. $rootScope - každá aplikace má jeden root scope. Všechny ostatní scope v aplikaci jsou potomci root scope. Root scope je přístupný všude v aplikaci

2. $scope - služba řeší kontext dat a je důležitý pro koncept data bindingu mezi HTML (view) a JavaScriptem (controller)

3. $http - služba, která usnadňuje komunikaci se vzdálenými HTTP servery prostřednictvím **XMLHttpRequest** objektu prohlížeče nebo prostřednictvím **JSONP**

### 4.1.3 Direktivy

Direktivy nám dávají možnost rozšířit HTML o nové možnosti. Direktivy jsou značky na DOM prvku (atribut, jméno elementu, CSS třída), které řeknou AngularJS compileru, že má přiřadit DOM prvku specifické chování (například přes události) nebo dokonce transformovat DOM prvek a jeho potomky. Framework obsahuje řadu vlastních předpřipravených direktiv, jejich seznam je dostupný v dokumentaci https://docs.angularjs.org/api/ng/directive.

```
angular.module('directiveTest', [])
.controller('Controller', ['$scope', function($scope) {
  $scope.customer = {
    name: 'Jaroslav',
    address: 'Ckalovova 41'
  };
}])
.directive('myCustomer', function() {
  return {
    restrict: 'E',
    template: 'Name: {{customer.name}} Address: {{customer.address}}'
  };
});
```

Výpis 5: Ukázka deklarace direktivy

```
<div ng-controller="Controller">
  <div my-customer></div>
</div>
```

Výpis 6: Ukázka použití direktivy v HTML

Na výpise 5 je ukázková deklarace vlastní direktivy. V deklaraci direktivy si můžeme všimnout atributu restrict, který říká, jaký DOM prvek ovlivní.

Restrict atribut můžeme nastavit na tyto hodnoty:

1. **A** - odpovídá pouze jménu atributu
2. **E** - odpovídá pouze jménu elementu
3. **C** - odpovídá pouze jménu css třídy
4. **M** - odpovídá pouze komentáři

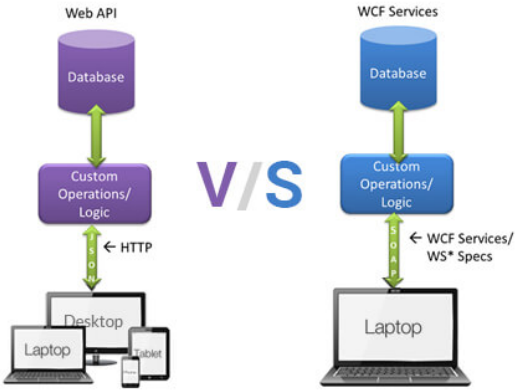
Restrikce na DOM prvky můžeme také kombinovat:

1. **AEC** - odpovídá buď jménu atributu, jménu elemtentu nebo jménu třídy

Dále si v ukázce můžeme všimnout atributu template, který obsahuje HTML šablonu, která se vyrenderuje do stránky, kde je direktiva použita. Pro definování HTML šablony můžeme také použít přímo HTML soubor, který je oddělen od kódu direktivy za použití atributu templateUrl místo template. Na výpise 6 je zobrazeno jak použít vlastní direktivu v HTML šabloně, která odpovídá jménu elementu. Toto je jen drobná ukázka jak lze pomocí direktiv rozšířit HTML možnosti, více o direktivách je k nalezení v dokumentaci https://docs.angularjs.org/guide/directive.

## 4.2 ASP.NET Web API

ASP.NET Web API je framework pro vytváření HTTP služeb, které mohou být konzumovány širokou škálou klientů, zahrnující internetové prohlížeče, mobilní telefony a tablety. Je velmi podobný ASP.NET MVC, neboť obsahuje funkce jako je routování, controlery, action resulty, filtry, model bindery a také Dependency Injection container. Avšak není součástí MVC Frameworku. Je součástí jádra ASP.NET platformy a může být použit s MVC a jinými typy webových aplikací jakou jsou ASP.NET WebForms. Může být také použit jakou samostatná webová služba.

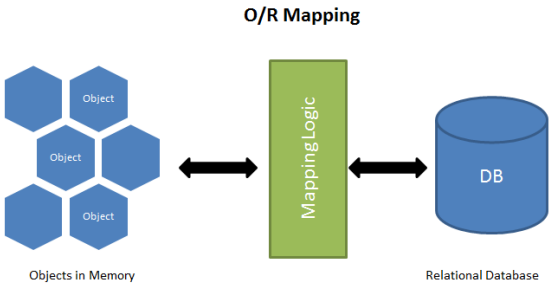



Obrázek 7: Rozdíl mezi Web API a WCF (zdroj : http://blog.andolasoft.com/2014/10/wcf-or-asp-net-web-api-which-one-to-choose.html

V dnešní době nejsou již webové aplikace postačující k uspokojení zákazníků. Lidé používají širokou škálu zařízení jakou jsou telefony, tablety apod. a ty mají k dispozi spousty aplikací. Pokud tedy chceme vystavovat data internetovým prohlížečů a také těmto moderním zařízením rychle a jednoduše, je potřeba mít API, které je kompatibilní s internetovými prohlížeči a různými zařízeními. Web API je výborný framework pro vystavování dat a služeb pro různá zařízení. Navíc Web API je open source a ideální platforma pro budování REST - ových služeb za pomocí .NET Frameworku. Na rozdíl od WCF Rest služby používá plno funkcí HTTP (jako jsou URI, request/response, hlavičky, caching, verzování, různé druhy formátů) a není nutné definovat žádné extra nastavení pro různá zařízení jako u WCF Rest služby.

## 4.3 Entity Framework

Objektově relační mapování je technika, která zajišťuje automatickou konverzi relačních dat (uložených v relační databázi) do prostředí objektově orientovaného programování. Způsobů jak realizovat samotné ORM je hned několik. Dokazují to návrhové vzory jako jsou Table Data Gateway, Row Table Gateway, Active Record a Data Mapper. Na obrázku 8 je jednoduchá ilustrace ORM.




Obrázek 8: Ilustrace ORM (zdroj : http://www.agile-code.com/blog/microsoft-net-or-mapper-choose-your-own/

Praktický přístup k ORM může být dvojí, můžeme vytvořit vlastní ORM, nebo můžeme použít již existující ORM framework. Oba tyto přístup mají své pro a proti.

- Vlastní implementace ORM
  - \+ plná kontrola nad SQL příkazy
  - \+ můžeme používat rysy specifické pro konkrétní SŘBD (datové typy, funkce, speciální dotazy atd.)
  - \+ při správné implementaci rychlejší
  - – doba strávená implementací ORM
  - – problematická změna používaného SŘBD.
- Použití existujících ORM frameworků
  - \+ rychlejší tvorba aplikace (vývojář se zabývá tvrobou business logiky namísto implementace ORM)
  - \+ jednoduchá změna používaného SŘBD
  - – pouze částečná kontrola nad generovanými SQL příkazy
  - – často nižší výkon

Při rozhodování mezi vlastní implementací ORM a použitím existujícího ORM nástroje je potřeba zvážit řadu faktů. Například jaké jsou požadavky na systém, kolik uživatelů bude se systémem pracovat atd.

V implementované aplikaci není očekávána nikterak veliká náročnost na systém ani velké množství uživatelů pracujících v aplikaci současně. Dále jsem se v aplikaci věnovat implementaci business logiky, proto byl pro vývoj použit existující ORM framework a to konkrétně Entity Framework, což je open source framework vyvinutý společnosti Microsoft. Entity Framework lze popsat následovně (definice převzata z anglického originálu dostupného na adrese [16])

*"Microsoft ADO.NET Entity Framework je framework pro Objektové/Relační Mapování (ORM), který umožňuje vývojářům pracovat s relačními daty jako s doménově-specifickými objekty. Eliminuje části kódu pro přístup k datům, které musí vývojář obvykle psát. Použitím Entity Frameworku mohou vývojáři používat pro dotazování LINQ, poté dostanou silně typové objekty, se kterými mohou dále manipulovat. Entity Framework implementace ORM poskytuje služby jako je change tracking (sledování změn), lazy loading a překlad dotazů, takže se vývojáři mohou soustředit pouze na specifickou business logiku vyvíjené aplikace namísto řešení základů přístupu k datům"*

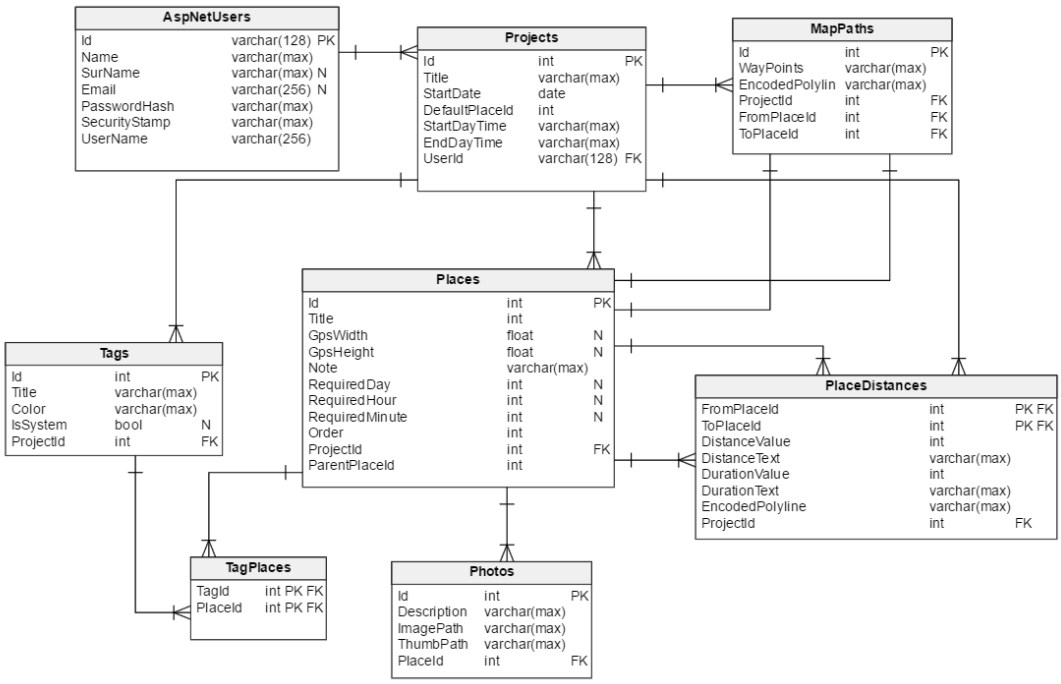
Entity Framework se používá ve třech scénářích. První scénář, pokud již databáze existuje nebo pokud chceme databázi navrhnout dříve než všechny ostatní části aplikace. Druhý, pokud se chceme zaměřit na doménu a doménové třídy a poté databázi vytvořit z doménových objektů. Třetí, pokud chceme schéma databáze navrhnout za pomocí vizuálního návrháře a poté ze schématu vytvořit databázi a třídy. V aplikaci jsem zvolil druhý přístup, tedy nejprve jsem si vytvořil doménové objekty a z těch následně vygeneroval databázi.

# 5 Návrh a implementace

Jedná se o webovou aplikaci tedy typu klient server. Aplikace se skládá ze tří základních vrstev, kde každá vrstva je zodpovědná za jinou úlohu v aplikaci. První vrstvou je prezentační vrstva, která má za úkol zobrazit data uživateli a zpřístupnit funkce aplikace. Další je aplikační vrstva neboli business layer, která je zodpovědná za vykonávání samotných výpočtů, rozhodnutí, nebo také autorizaci či autentizaci. Nejnižší vrstva je datová, která má za úkol poskytovat data aplikační vrstvě a také data perzistovat do datového úložiště. V této konkrétní aplikaci je to databáze.

## 5.1 Vrstva pro přístup k datům

Jako úložiště pro data byla zvolena databáze, konkrétně je použita databáze MSSQL 2014. Pro přístup k datům byl použit ORM framework Entity Framework verze 6. Na obrázku 9 je zobrazeno schéma použité databáze.




Obrázek 9: Schéma databáze

V Aplikaci je potřeba často pracovat s daty uloženými v databázi, proto jsem vytvořil modul, který vystavuje rozhraní pro práci s databází. Na obrákzu 10 je třídní diagram modulu. Jedním ze základních pilířů modulu je "generický repozitář". Tento "repozitář"poskytuje jednotné rozhraní pro základní CRUD operace.

Pokud potřebujeme složitější dotaz do databáze než-li je výpis všech entit z tabulky, je potřeba použít takzvaný "query objekt". Všechny složitější dotazy v aplikaci mají svou vlastní

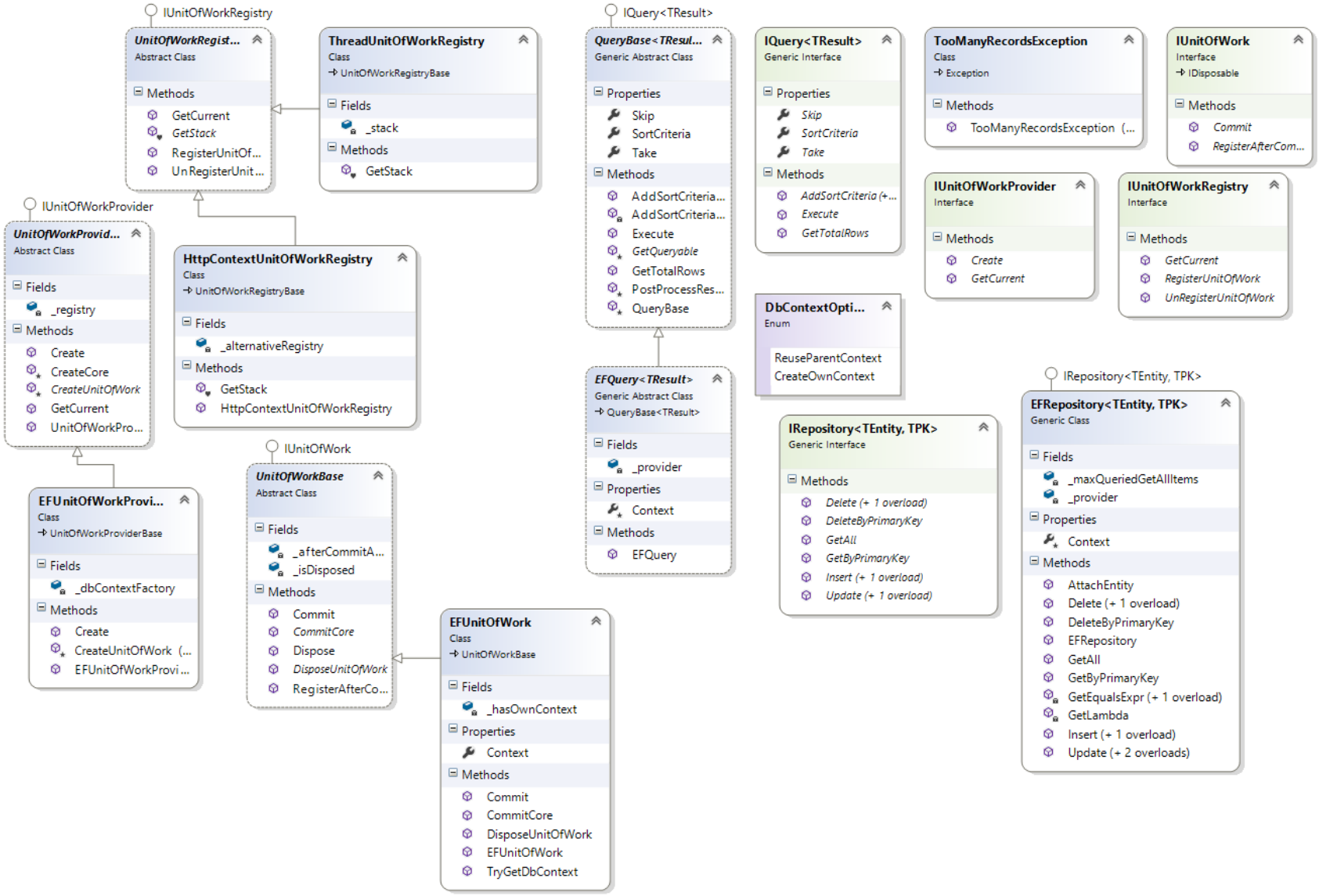
třídu, která je potomkem generické třídy "EFQuery". Základní myšlenkou těchto objektů je implemenovat metodu "GetQueryable", která bude obsahovat samotný dotaz, a samotné spuštění dotazu se provede pomocí metody "Execute", kterou konkrétní třída dědí z bázové abstraktní třídy. Metoda "Execute"je reálný příklad použití návrhového vzoru Template Method.

```
public IList<TResult> Execute()
{
    var query = GetQueryable();
    for (int i = SortCriteria.Count - 1; i >= 0; i--)
        query = SortCriteria[i](query);

    if (Skip.HasValue)
        query = query.Skip(Skip.Value);
    if (Take.HasValue)
        query = query.Take(Take.Value);

    var result = query.ToList();
    PostProcessResult(result);

    return result;
}
```

Výpis 7: Metoda Execute abstraktní třídy QueryBase

Na výpise 7 si lze také všimnout, že metoda pro vykonání dotazu podporuje také stránkování, pokud jsou vlastnosti "Skip"a "Take"nastaveny. Dále je po vykonání dotazu ještě volána metoda "PostProcessResult", která provede operaci na vrácených datech, pokud metodu implementujeme.

Další časti modulu jsou třídy, které implementují návrhový vzor Unit of Work. Základními stavebními kameny jsou třídy "UnitOfWorkBase"a "UnitOfWorkProvierBase". Třída "UnitOfWorkProviderBase"a její potomci jsou zodpovědní za vytvoření objektu "UnitOfWorkBase"nebo jejího potomka. Objekty "UnitOfWork"definují rozhraní, které obsahuje dvě metody. První je "Commit"pro zapsání všech změn provedených v paměti do datového úložiště. A druhou je "RegisterAfterCommitAction"pro registrování dalších akcí, které se mají provést po metodě "Commit".




Obrázek 10: Třídní diagram core modulu pro práci s daty

```
public IEnumerable<Place> GetAllAvailablePlaces(int projectId, string userName,
     int currentPlaceId)
{
    var filter = new PlaceFilter { ProjectId = projectId, UserName = userName,
       CurrentPlaceId = currentPlaceId };
    using (var uow = _provider.Create())
    {
        var query = new AllAvailablePlacesQuery(_provider) { Filter = filter };

        return query.Execute();
    }
}
```

Výpis 8: Ukázka použití UnitOfWork spolu s query objektem

Na výpise 8 je ukázka, jak vypadá použití těchto objektů v praxi. Nejdříve si připravíme objekt s hodnotami, které chceme z tabulky "Places"filtrovat. Následně pomocí instance třídy "EFUnitOfWorkProvider"vytvoříme objekt "EFUnitOfWork", který celou operaci uzavře do jednoho logického celku. Dále vytvoříme konkrétní instanci query objektu a provedeme metodu "Execute", která vrátí výsledek dotazu a data jsou předány vyšší vrstvě.

Jelikož používám pro přístup k datům Entity Framework, dotazy do databáze nepíši přímo v SQL, ale používám "LINQ to Entities". Dotazy jsou frameworkem vygenerovány do dotazovacího jazyka SQL automaticky.

```
return Context.Photo.Join(Context.Place,
                          photo => photo.PlaceId,
                          place => place.Id,
                          (photo, place) => new { ProjectId = place.ProjectId,
                             Photo = photo })
                    .Where(placeAndPhoto => placeAndPhoto.ProjectId == Filter.
                       ProjectId)
                    .GroupBy(placeAndPhoto => placeAndPhoto.Photo.PlaceId)
                    .Where(photoGroup => photoGroup.Count() > 0)
                    .Select(photoGroup => photoGroup.OrderByDescending(p => p.
                       Photo.Id).FirstOrDefault())
                    .Select(photo => photo.Photo);
```

Výpis 9: Ukázka definice dotazu v aplikaci

```
SELECT [t5].[Id], [t5].[ImagePath] FROM ( SELECT COUNT(*) AS [value], [t0].[
    PlaceId]
    FROM [Photos] AS [t0]
    INNER JOIN [Places] AS [t1] ON [t0].[PlaceId] = [t1].[Id]
    WHERE [t1].[ProjectId] = @p0
    GROUP BY [t0].[PlaceId]
```

```
    ) AS [t2]
OUTER APPLY (
    SELECT TOP (1) [t3].[Id], [t3].[ImagePath] FROM [Photos] AS [t3]
    INNER JOIN [Places] AS [t4] ON [t3].[PlaceId] = [t4].[Id]
    WHERE ([t2].[PlaceId] = [t3].[PlaceId]) AND ([t4].[ProjectId] = @p0)
    ORDER BY [t3].[Id] DESC
    ) AS [t5]
WHERE [t2].[value] > @p1
ORDER BY [t5].[Id] DESC
```

Výpis 10: Vygenerovaný SQL dotaz

Na výpise 9 je ukázka, jak vypadá takový dotaz zapsaný pomocí "LINQ Extenstions", a na výpise 10 je vygenerovaný SQL dotaz Entity Frameworkem. V dotazu je vynechána část sloupců, které jsou vybírany.

## 5.2 Business vrstva

V aplikaci hlavním úkolem business vrstvy je výpočet optimální trasy pro uživatelem zadaná místa a vybraný biologicky inspirovaný algoritmus. Další nedílnou části business vrstvy je zjišťování vzdáleností mezi jednotlivými místy, které jsou potřeba pro výpočetní algoritmy.

### 5.2.1 Získání vzdáleností

Pro získávání vzdáleností mezi jednotlivými místy, které si uživatel uloží, je použita služba od firmy Google. Firma Google nabízí zdarma širokou škálu webových služeb podporující práci s mapou. Pro získávání vzdáleností mezi místy je použito Distance Matrix API, dokumentace k tomuto api je dostupná na adrese https://developers.google.com/maps/documentation/distance-matrix/intro. Pro práci se službami je nutné vlastnit tzv. *API KEY*, který lze získat na adrese https://console.developers.google.com/apis/credentials?project={Název projektu}, nebo na stránkách dokumentace dané služby, kde je většinou tlačítko pro získání klíče.

Jelikož je api zdarma, obsahuje nějaká omezení, která je nutné brát v potaz. Každý dotaz, který je poslán na Distance Matrix API, je omezen počtem povolených elementů, kde počet *origins* krát počet *destinations* definuje počet elementů. Pro standardní použití je počet elementů omezen na 2500 za den, v prémiové verzi je počet elementů omezen na 100000 za den. Dále je omezen maximální počet *origins* a *destinations* na 25 na požadavek v obou variantách. Limity jsou dostupné na adrese https://developers.google.com/maps/documentation/distance-matrix/usage-limits

Api dokáže vracet dva formáty a to JSON a nebo XML. V aplikaci jsem zvolil formát JSON. Na výpise 11 je ukázka odpovědi z api pro požadavek `https://maps.googleapis.com/maps/api/distancematrix/json?origins=49.9361958596793,17.9042816162109&destinations=50.079010595097,14.4330739974976&key=YOUR_API_KEY`

```
{
   "destination_addresses" :["Vinohradska 343/6, Vinohrady, 12000Praha-Praha 2,
      Cesko"],
   "origin_addresses" :["U Posty 251/8, Mesto, 74601Opava, Cesko"],
   "rows" :[
      {
         "elements" :[
            {
               "distance" :{
                  "text" :"370 km",
                  "value" :370270
               },
               "duration" :{
                  "text" :"3 hod, 41min",
                  "value" :13251
               },
               "status" :"OK"
            }
         ]
      }
   ],
   "status" :"OK"
}
```

Výpis 11: Odpověď z Distance Matrix API ve formátu JSON

Do adresy požadavku lze také přidat parametr *mode*, který specifikuje druh dopravy, která je použita pro kalkulaci vzdálenosti. Vzdálenosti jsou získány při zahájení vyhledávání optimální trasy pro místa. Pro každé místo, které je zahrnuto do vyhledávání, se nejdříve zkontroluje, do kterých míst chybí vypočítana vzdálenost, a vytvoří dynamicky požadavek, kde v parametru *destinations* budou odděleny jednotlivé souřadnice míst. Odpověď je deserializována do objektů a ty jsou následně uloženy do databáze.

### 5.2.2 Získání trasy mezi místy

Pro získání dat, která jsou potřeba pro vykreslení cest mezi jednotlivými místy, je opět použita služba od firmy Google. Jedná se o Directions API, dokumentace k api a podrobnější význam použitelných parametrů je dostupná na adrese `https://developers.google.com/maps/documentation/directions/intro`.

```
{
   /* Zbytek odpovedi */

         "overview_polyline" :{
            "points" :"gzapHavriBOjB .... ~HvD|Ah@p@HJkFgBg@_@SQ?IRMX"
         },
         "summary" :"silnici 11",
         "warnings" :[],
         "waypoint_order" :[]
      }
   ],
   "status" :"OK"
}
```

Výpis 12: Odpověď z Directions API ve formátu JSON

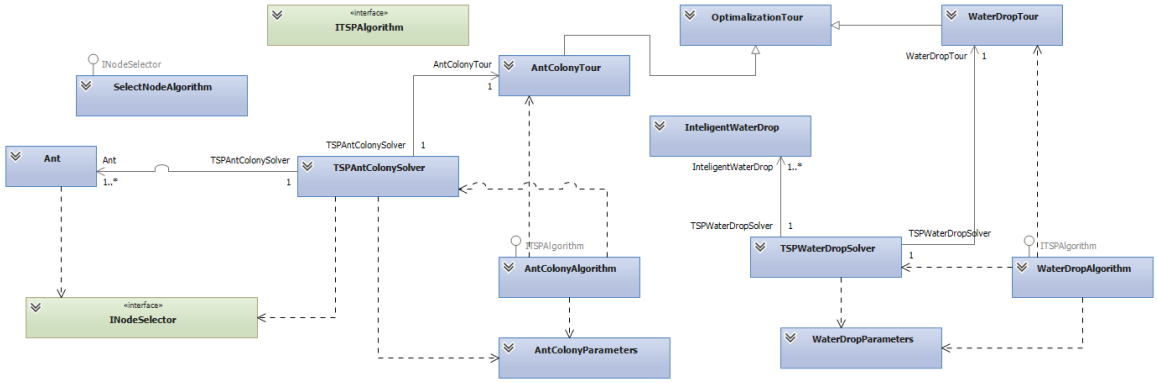
Na 12 je ukázka části odpovědi z Directions API pro požadavek `https://maps.googleapis.com/maps/api/directions/json?origin=49.98611063781005,17.46826171875&destination=49.93177612998565,17.900848388671875&sensor=false&mode=driving&key=YOUR_API_KEY`. Důležitou částí pro vykreslení cesty je hodnota vlastnosti "points", který obsahuje zakódované souřadnice cesty mezi místy. Popis algoritmu, který je použit pro zakódování souřadnic cesty, je dostupný na adrese `https://developers.google.com/maps/documentation/utilities/polylinealgorithm`.

Pro získání trasy mezi optimalizovanými místy lze využít dva způsoby. První, pro každou cestu mezi místy zavoláme Directions API zvlášť. Nebo použijeme tzv. *waypoints*, které definují místa, přes která chci projet při cestě z místa A do místa B. Parametr *waypoints* je omezen maximálním počtem 23 míst. V době, kdy jsem tuto část aplikace psal, byl dokonce maximální počet 8 míst. V práci jsem zvolil první přístup, tedy pro každou trasu mezi dvěma místy volám api. Není to optimální, protože velice rychle bude vyčerpán denní limit requestů, který je 2500. Nicméně v aplikaci neočekávám tisíce uživatelů, kteří by chtěli optimalizovat trasy s 50 a více místy, proto jsem ponechal první řešení. Toto řešení jsem také ponechal z důvodu nedostatku času, protože bych musel předělávat více částí v aplikaci. Získaná data se ukládají do databáze, alespoň tímto je ušetřeno opakovaných volání api. V aplikaci je vytvořena také implementace posílání požadavků s parametrem *waypoints* a následné zpracování a uložení do databáze, tato implementace se dá velice jednoduše nahradit tou aktuálně použitou. Nicméně zpracování vlastnosti "points"z odpovědi je o něco náročnější, protože ve finálním shruntí není cesta při vykreslení do mapy přesná. V odpovědi je více jednotlivých úseků, které mají svoji vlastnost "points"a muselo by se přepsat zpracování odpovědi, aby byl zobrazený výsledek kvalitnější.

### 5.2.3 Optimalizační algoritmy

Nejdůležitější úlohou business vrstvy v aplikaci je najít optimální trasu místy, která zvolí uživatel, a výsledek předat prezentační vrstvě, která jej adekvátně zobrazí uživateli. Na obrázku

11 je zobrazen třídní diagram obou implementovaných algoritmů. Důležitou abstrakcí je zde interface "ITSPAlgorithm", který je potřeba z důvodu, že uživatel má možnost si libovolně měnit (v uživatelském rozhraní) algoritmus výpočtu, a je tedy nutné, aby určité části aplikace nebyly přímo závislé na algoritmu ale pouze na abstrakci.




Obrázek 11: Zjednodušený třídní diagram optimalizačních algoritmů

### 5.2.4 Implementace Ant Colony algoritmu

Základním prvkem algoritmu je simulovaný mravenec, který provádí průchod grafem. Simulovaný mravenec je reprezentován třídou "Ant", která obsahuje jedinou metodu "ConstructTour", ve které mravenec projde graf problému. Každý simulovaný mravenec má informaci o navštívených místech uloženou v proměnné *visitedNodes*. Nejprve do seznamu přidáme startovací místo, a poté dokud, mravenec nenavštíví všechna místa, je volána metoda "SelectNextNodeFromNode", která vrací následující místo z aktuálního místa mravence. Metoda implementuje algoritmus pro výběr místa tak, jak je popsáno v kapitole 3.2.3. Metoda dále navyšuje vzdálenost, kterou mravenec urazil, pro pozdější vyhodnocení všech nalezených cest všemi mravenci. Na výpise 13 je zobrazena metoda, která je implementací rozhraní "ITSPAlgorithm". Metoda obsahuje části, které jsou popsány v kapitole 3.2.2. Každá iterace algoritmu začíná metodou "AntColonyIteration", která rozmístí náhodně simulované mravence po grafu reprezentujícím problém. Následně všichni simulovaní mravenci zkonstruují svou cestu grafem. Jakmile všichni mravenci dokončí své cesty, je zavolána metoda "PheromoneEvaporation", která provede vypařování feromonů. Následně metoda "AddNewPheromones"vypočítá nové přírustky feromonů na jednotlivých cestách mez místy. Dále je provedena metoda KeepBestSolution, která porovná všechny cesty provedené mravenci a zapamatuje si tu nejlepší, pokud v aktuální iteraci byla nalezena vhodnější než v předešlé. Nakonec iterace je provedena metoda "ReinforceBestTour", která na doposud nejvhodnější cestě zvýší hodnotu feromonu. Hledání trasy končí po dosažení maximálního počtu iterací.

```
public OptimalizationResult Compute()
{
    // Omited code

    for (int iteration = 1; iteration ≤_tspSolver.Parameters.IterationsCount;
        iteration++)
    {
        _tspSolver.AntColonyIteration();
        _tspSolver.PheromoneEvaporation();
        _tspSolver.AddNewPheromones();
        _tspSolver.KeepBestSolution();
        _tspSolver.ReinforceBestTour();
    }

    //Omited code
}
```

Výpis 13: Metoda Compute AntColonyAlgorithm třídy

### 5.2.5 Implementace Water Drop algoritmu

Základním prvkem algoritmu je Inteligent Water Drop, která je reprezentována třídou "InteligentWaterDrop". Třída obsahuje důležitou metodu "NextNodeIdBasedOnSoil", která vrací místo, do kterého se IWD přesune z aktuálního místa. Na výpise 14 jsou části metody zobrazeny. Metoda implementuje rovnice pro získání pravděpodobnosti výběru místa z kapitoly 3.3.2. Nejprve si zjistíme, jaká je minimální hodnota parametru *soil* na cestách mezi místy. Poté pro IWD získáme všechna místa, která ještě nenavštívila, a vypočítáme hodnotu *fsoilSum*. Po získání nezbytných hodnot jsou vypočítány pravděpodobnosti pro návštěvu všech doposud nenavštívených míst. Po vybrání největší hodnoty je určené následné místo uloženo do seznamu navštívených míst a vráceno dále ke zpracování.

Metoda "Compute"třídy "WaterDropAlgorithm", která je implementací rozhraní "ITSPAlgorithm", je velice obdobná pro Ant Colony algoritmus. Každá iterace algoritmu začíná metodou "WaterDropsIteration", ve které všechny IWD zkonstruují cestu. Následně je uloženo nejlepší řešení, tedy pokud bylo nějaké lepší než v předešlé iteraci, a reinicializují se hodnoty jednotlivých IWD. Pokračuje se další iterací, dokud není překročen maximální počet iterací.

```
public int NextNodeIdBasedOnSoil(Tour tour)
{
    if (VisitedNodesCount == tour.Nodes.Count)
        return -1;
    // Omited code
    foreach (var entryNode in unvisitedNodes)
    {
        if (!choosingPropability.ContainsKey(entryNode.Value.ID))
        {
            var probability = tour.GetF_Soil(CurrentNodeId, entryNode.Value.ID,
                minimumSoil) / fsoilSum;
            choosingPropability.Add(entryNode.Value.ID, probability);
        }
    }

    nextNodeId = choosingPropability.FirstOrDefault(x => x.Value ==
        choosingPropability.Values.Max()).Key;
    // Omited code
}
```

Výpis 14: Metoda NextNodeIdBasedOnSoil InteligentWaterDrop třídy

## 5.3 Prezentační vrstva

Hlavním úkolem prezentační vrsty je adekvátně uživateli aplikace zobrazit data a zpřístupnit uživateli funkce systému. Jedná se o webovou aplikaci, klientská část tedy běží na straně internetového prohlížeče, přes který uživatel interaguje se systémem. Základními funkcemi tohoto systému je správa míst a spuštění optimalizace trasy a vykreslení optimalizované trasy v mapě.

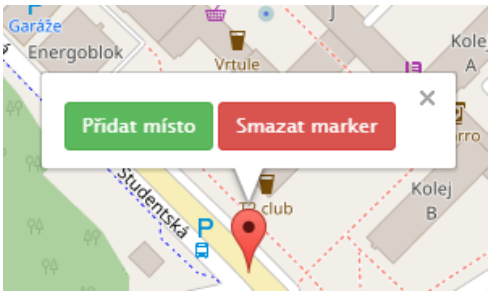
### 5.3.1 Správa míst

Základní datovou informací jsou data o místech, která chce uživatel navštívit. Hlavní myšlenkou pro práci s místy je jednotlivá místa přiřazovat pod projekty. Uživatel si tedy nejprve vytvoří jednotlivé projekty (při registraci je každému uživateli vygenerován defaultní projekt) a následně pro aktuálně vybraný projekt vytváří místa. Projekt by zde měl zastupovat logický celek, který sdružuje místa, která jsou optimalizována pro ideální průchod.

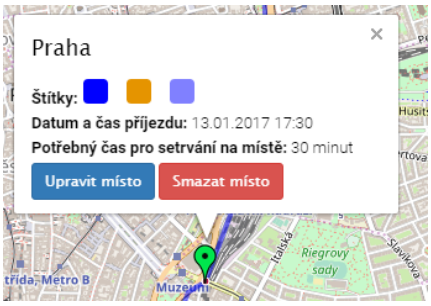
Základní informací o místě, které chce uživatel navštívit, jsou GPS souřadnice. Pro větší komfort pro přídávání míst lze využít na stránce s mapou vyhledávacího pole, které při vyhledání přiblíží mapu na vyhledané místo. Uživatel tak snadno může na mapě kliknutím označit přesnou adresu a přes kontextové menu přidaného markeru se může přesunout na formulář pro vytvoření místa, kde budou předvyplněná pole s GPS souřadnicemi.

Pro sdílení hodnot mezi jednotlivými controllery je použita služba "SharedPropertiesService". Služba obsahuje dvě funkce a to "setProperty"a "getProperty". Funkce "setProperty"očekává na vstupu název klíče, pod kterým bude druhý paremetr objektu, který chceme uložit, uložen.

Funkce "getProperty"podle klíče vrátí hodnotu. Při přesunu z kontextového menu markeru mapy jsou pomocí služby přídány hodnoty nadmořské výšky a šířky. V controlleru obsluhující vytvoření nového místa opět pomocí služby jsou tyto hodnoty nalezeny, pokud existují, a přiřazeny polím na formuláři.




Obrázek 12: Kontextový dialog markeru




Obrázek 13: Kontextový dialog uloženého místa

Na obrázku 12 je zobrazen kontextový dialog markeru, který uživatel při kliknutí na mapě vytvoří. Na obrázku 13 je zobrazeno kontextové menu místa, které již uživatel má uloženo v systémové databázi. Dialog zobrazuje informace, které jsou vyplněny při ukládání místa. Pokud si uživatel uloží optimalizovanou trasu, v dialogu se také zobrazí informace o času příjezdu. Barevné označení míst určuje, zda je místo uloženo v databázi (zelená barva), nebo jestli se jedná pouze o dočasné označení (červené barva) GPS souřadnic na mapě. Nedílnou součástí uživatelského rozhraní je také stránka se seznamem všech uložených míst. Navíc, pokud je již uložena optimalizovaná trasa, jsou místa seřazena a mezi místy je zobrazena informace o době cesty a délce trasy.

Další zajímavá funkcionalita spojená se správou míst je možnost přidávat fotografie k jednotlivým místům. Každému místu můžeme vytvořit fotogalerii. Ve fotogalerii jsou zobrazeny miniatury fotografií, kdy při najetí na miniaturu má uživatel možnost fotografii smazat. Při kliknutí na fotografii se zapne mód prohlížení. Fotogalerie obsahuje tlačítko pro přidání fotografií, které umožňuje vybrat fotografie ze souborového systému počítače a nahrát je na server. Jakmile je soubor vybrán, je okamžitě odeslán na server a tam dále zpracován. Dialogové okno umožňuje výběr více fotografií naráz. Při odesílání fotografií je uživateli zobrazen progress bar se stavem odeslaných dat. O toto chování se stará direktiva "ngfSelect", která je dostupná na adrese `https://github.com/danialfarid/ng-file-upload`.

Na výpise 15 je zobrazeno použití direktivy. V parametru *accept* určujeme jaký typ souboru lze nahrát na server. Parametrem *ngf-max-size* určujeme maximální velikost nahrávaného souboru, která je nastavena na 5 MB. Samotné odesílání fotografií je provedeno ve funkci "uploadFiles", která postupně odešle soubory na server.

```
<button ngf-select="uploadFiles($files, $invalidFiles)" multiple accept="
    image/*" ngf-max-size="5MB" class="btn btn-primary btn-sm mb10" type="
    button" />
```

Výpis 15: Použití direktivy ngfSelect v HTML

Jakmile server převezme požadavek na uložení souboru, vygeneruje pro něj "Guid"jako název souboru, aby byla zajištěna unikátnost názvu souboru. O vygenerování takového názvu se stará overridenutá metoda "GetLocalFileName"třídy "MultipartFormDataStreamProvider", která dědí ze třídy "MultipartFormDataStreamProvider". Tento provider je použit v metodě pro čtení všech částí MIME vícedílné zprávy z příchozího požadavku, která vrátí data nahrávaných fotografií. Po získání dat je pomocí třídy "ImageSaver"vytvořena miniatura fotografie a následně jsou informace o fotografii uloženy do databáze. V databázi jsou uloženy pouze cesty k jednotlivým fotografiím.

### 5.3.2 Práce s mapou

Hlavním úkolem systému je optimalizovat průchod místy a také je potřeba výsledek uživateli adekvátně zobrazit. Pro práci s mapou na klientské části byla využita dostupná Google Maps JavaScript API. Pro práci s mapou byla použita direktiva "uiMap"z modulu "UI.Map", která je volně dostupná ke stažení na Githubu na adrese `https://github.com/angular-ui/ui-map`. Direktiva umožňuje přidávat do stránky Google Maps JavaScript API elementy. Veškeré funkce, které jsou spojeny s mapou, jsou součástí "MapControlleru".

```
<div id="map_canvas" ui-map="model.myMap" class="map"
     ui-event="{'map-click': 'addMarker($event, $params)' }" ui-options="
        mapOptions">
</div>
```

Výpis 16: Ukázka použití direktivy ui-map v HTML

Na výpise 16 je ukázka použití direktivy. Direktiva se sama postará o vytvoření objektu mapy, který je součástí Google Maps JavaScript API a následné přiřazení do modelu **model.myMap**. Direktiva se také postará o nabindování událostí jako například *map-click* na objekt mapy. Pro registrování události, která se má provést po kliknutí na mapu, je použita direktiva "UI.Event", která umožňuje nabindovat callback na jakoukoli událost, která není nativně podporována frameworkem AngularJS. Funkce "addMarker", která je registrována na událost kliknutí na mapu, přidává objekt typu "google.maps.Marker"do pole všech markerů, které jsou zobrazeny na mapě. Parametr *ui-options* slouží pro nastavení vlastností mapy jako například nastavení výchozího typu mapy, výchozí hodnota zoomu atd.

Modul "UI.Map"obsahuje také direktivu "uiMapInfoWindow", která slouží pro zobrazení kontextového okna po kliku na marker. Na výpise 17 je ukázka použití direktivy. Direktiva se stará o vytvoření objektu "google.maps.InfoWindow", který je přiřazen do modelu **myInfoWindow**. Pomocí direktiv "ng-repeat","ui-event"a "ui-map-marker"je namapována na všechny objkety typu marker událost, která zavolá funkci "openPlaceInfo"s parametrem *place*. Funkce pomocí objektu "myInfoWindow"otevře kontextové menu, které je zobrazeno na obrázku 13.

```
<div ng-repeat="place in mapPlaces" ui-map-marker="mapPlaces[$index]"
    ui-event="{'map-click': 'openPlaceInfo(place)' }">
</div>
<div ui-map-info-window="myInfoWindow" id="mapInfoWindow">
    .....
</div>
```

Výpis 17: Ukázka použití direktivy ui-map-info-window v HTML

Pro práci s mapou lze využít různých druhů mapy. K dispozici jsou tyto typy:

- RoadMap - zobrazuje klasickou silniční mapu
- Terénní - zobazuje fyzickou mapu na základě informací o terénu
- Satelitní - zobrazuje Google Earth satelitní snímky
- OpenStreetMap - zobrazuje speciální mapovou vrstu, která je tvořena komunitou uživatelů, kteří udržují data o silnicích, cestách, kavárnách, železničních stanicích a mnohém dalším po celém světě. Více informací o projektu je na adrese `https://www.openstreetmap.org/about`. Tato vrstva je nastavena jako výchozí

Pro vykreslení cesty na mapě je použit způsob vykreslování pomocí jednoduchých cest na mapě zvaných "Polylines". Nejprve jsou získána všechna uložená místa pro aktuálně vybraný projekt a přidána na mapu jako markery. Dále jsou získány ze serveru všechny dekódované souřadnice mezi jednotlivými místy. Na výpise 18 jsou zobrazeny části funkce, která se stará o vykreslení trasy. Ze souřadnic mezi jednotlivými místy je vytvořen seznam s objekty "google.maps.LatLng", které vytváří cestu mezi místy. Tyto objekty obsahující souřadnice jsou postupně vykreslovány na mapu pomocí objektu "google.maps.Polyline". Cesta mezi místy je na mapě znázorněna modrou barvou. Seznam se souřadnicemi obsahuje na prvním a posledním indexu souřadnice, které označují místa, pro které je vykreslována cesta. Pro tyto souřadnice je vykreslen na mapu červený kruh.

```
$scope.renderBestTour = function (polyLines) {
  //Omited code
    for (var lineKey in polyLines) {
```

```
        var line = polyLines[lineKey];
          angular.forEach(line, function (key, value) {
           flightPlanCoordinates.push(new google.maps.LatLng(key.Latitude, key.
               Longitude));
          });
          var route = new google.maps.Polyline({
               map: $scope.model.myMap,
               path: flightPlanCoordinates,
               strokeWeight: 4,
               strokeOpacity: 0.5,
               strokeColor: "blue"
              });
   //Omited code
   }
}
```

Výpis 18: Funkce pro vykreslení trasy

Všechny objekty, které jsou použity pro vykreslení na mapu, obsahují vlastnost "map", která definuje, na jaké mapě se objekty mají vykreslit. Při vykreslování trasy na mapu je nutné před začátkem vykreslování nové trasy nejprve projít všechny objekty, které jsou použity pro vykreslení trasy, a nastavit jim vlastnost "map"na hodnotu *null*.

Pro vyhledávání míst je použita direktiva "googlePlaces", která z textového pole vytvoří objekt "google.maps.places.Autocomplete"a asociuje objektu funkci na událost *place_changed*. Jakmile uživatel začne do textového pole psát znaky, automaticky budou našeptávána místa. Direktiva obsahuje jeden parametr *location*, do kterého je přiřazen model **location**, který je naplněn souřadnicemi vybraného místa. Po stisku tlačítka "Hledat", které je vedle textového pole, se vyvolá funkce "search", která vycentruje mapu na souřadnice vyhledávaného místa a nastaví zoom.

Po získání optimalizované trasy je také zobrazen panel s informacemi o trase. Panel obsahuje hlavičku se souhrnem cesty, která se skládá z celkového potřebného času na projetí trasy, dále obsahuje informaci o celkové vzdálenosti trasy. Hlavička obsahuje také datum a čas začátku a konce cesty. Datum a čas začátku cesty je nastaven v nastavení projektu. Datum a čas konce cesty je vypočítán z doby trvání průjezdu místy a dále je zahrnut do výpočtu i čas, který je potřebný pro navštívení místa. Panel obsahuje informace o délce trasy a potřebného času pro ujetí trasy mezi jednotlivými místy. Pro vykreslení panelu je použita direktiva "routeSteps". HTML template pro tuto direktivu je připravena ovšem na serveru metodou "GetRouteStepsHTML"třídy "RouteSteps".

```
link: function (scope, element, attrs) {
   if(scope.places.length) {
      AntColonyService.GetSteps(scope.places).then(function (data) {
         element.html(data).show();
         $compile(element.contents())(scope);
      }, function (data) {
         console.log("routeSteps error");
      });
   }
     scope.$watch('template', function () {
         element.html(scope.template);
         $compile(element.contents())(scope);
     });
}
```

Výpis 19: Funkce link direktivy RouteSteps

Na výpise 19 je zobrazena "link"funkce direktivy. Direktiva má svůj vlastní scope, který obsahuje řadu parametrů a jedním z nich je *template*. Při inicializaci direktivy je "link"funkce spuštěna, pokud je parametr *places* naplněn, je voláno serverové API, které vrátí připravenou HTML šablonu. Šablona je vložena do parametru *element* pomocí funkce "html". Parametr *element* obaluje element, na kterém je direktiva aplikována jako jQLite objekt. jQLite je drobná podmnožina jQuery, která umožňuje frameworku AngluraJS provádět manipulace s DOMem kompatibilně napříč všemi prohlížeči. Na adrese `https://docs.angularjs.org/api/ng/function/angular.element` je seznam podporovaných funkcí. Pokud je jQuery definováno v HTML dříve než AngularJS, je elemenet obalen jQuery namísto jQLite. Jakmile je element obalen pomocí jQLite/jQuery, není potřeba element obalovat $(), abychom mohli provádět DOM manipulace. Jakmile je šablona vložena do elementu, je pomocí služby $compile view zkompilováno a nabindováno na scope, tímto je zajištěno správné zobrazení HTML obsahu.

Dále je v "link"funkci definována **$watch** funkce na model **template**. Díky této funkci, jakmile je na modelu provedena nějaká změna, například přiřazení nové hodnoty, je spuštěna funkce, která je definovaná jako druhý argument. Díky této funkci je možné po optimalizaci trasy, kdy je ze serveru vrácena nová šablona, tuto novou šablonu uživateli korektně zobrazit.

### 5.3.3 Kalendář událostí

Pokud si uživatel uloží optimalizovanou trasu, na stránce s kalendářem se mu zobrazí jednotlivé události v kalendáři. Návštěva místa je označena modrou barvou a cesta mezi jednotlivými místy červenou. Pro práci s kalendářem je použita AngularJS direktiva "ui-calendar"(dostupná na adrese `https://github.com/angular-ui/ui-calendar`), která obaluje práci s knihovnou

"FullCalendar"z jQuery. Direktiva obsahuje model **eventSources**, který je určen pro kolekce událostí, které mají být vykresleny v kalendáři. Direktiva sleduje změny na tomto modelu. Direktiva obsahuje také parametr "ui-config", do kterého je předáno nastavení pro kalendář, které je zpracováno knihovnou "FullCalendar".

V konfiguračním parametru lze také nastavit, jaká funkce se má provést při kliknutí na událost. Této možnosti je využito a při kliknutí na událost v kalendáři, která představuje dobu setrvání na místě, je zobrazen dialog s informacemi o místě. Pro zobrazení dialogu je použita služba **$uibModal**, která je součástí modulu "ui.bootstrap". Dokumentace k modulu je dostupná na adrese `https://angular-ui.github.io/bootstrap/`. Vytvoření modálního okna je přímočaré, je potřeba vytvořit HTML template a controller a při vytváření modálního okna toto použít. Služba **$uibModal** obsahuje jedinou funkci a tou je funkce "open", která očekává na vstupu objekt nastavení modálního okna. Funkce vrátí instanci modálního okna, která obsahuje řadu vlastností. Jednou z těchto vlastností je funkce "dismiss", která modální okno zavře.

Při vytváření modálního okna je v objektu nastavení použita vlastnost "resolve". Tato vlastnost obsahuje objekt s vlastnostmi, které jsou resolvovány a poslány jako parametry controlleru. Díky téhle možnosti je možné z controlleru kalendáře poslat do controlleru modálního okna objekty aktuálního místa, následujícího místa a vzdálenosti mezi těmito místy. Informace těchto objektů jsou prezentovány uživateli v modálním okně. Dále je zde možnost přesunout se na editaci vybraného místa.

### 5.3.4 Informační hlášky

Uživateli jsou zobrazovány informační hlášky během práce s aplikací, které se skládají z nadpisu a textu. Například při úspěšném uložení enity nebo při úpravě entity. Pro zobrazování informačních hlášek je použit modul "toaster". Modul obsahuje službu "toaster", která obsahuje jedinou funkci "pop". Funkce vytvoří objekt informační hlášky a pomocí broadcastu vyvolá funkci "addToast"direktivy "toasterContainer", která je také součástí modulu. Direktiva definuje základní funkci "addToast", která jako argument přijímá objekt hlášky. Objekt obsahuje řadu vlastností, které jsou ve funkci zpracovány, například typ hlášky (info, error, success, warning), podle kterého je určena css třída, která je aplikována. Objekt také obsahuje vlastnost, za jak dlouhý časový interval hláška zmizí. Direktiva registruje posluchače na událost "toaster-newToast", díky tomu, když je vyvolán broadcast s touto událostí, je funkce "addToast"spuštěna.

Direktiva obsahuje řadu parametrů, které ovlivňují chování a vzhled jednotlivých hlášek. Například jdou nastavit css třídy pro jednotlivé typy hlášek, css třídu pro pozici hlášek, css třídy pro titulek a samotnou zprávu. Direktiva vykresluje jednotlivé hlášky pomocí direktivy "ng-repat", která je definovaná v template direktivy.

Pro zpracování chybových hlášek je zapotřebí zapojit serverovou i klientskou část aplikace. Na serverové straně bylo potřeba napsat globální handler, který odchytává výjimky. Pro odchytávání výjimek je použita třída "GlobalExceptionHandler", která dědí z třídy "ExceptionHandler"a přepisuje metodu "Handle". Metoda vytváří odpověď se statusem 500 (Internal Server Error) a

přidává zprávu z výjimky. Tato odpověď je vrácena klientské části. Aby Web API vědělo, že v případě výjimky ma použít speciální handler, je nutné při konfiguraci služeb pro Web API defaultní handler pro odchytávání chyb nahradit handlerem, který chceme použít.

Na klientské části je pro globální zpracování a zobrazování chybových hlášek využit tzv. interceptor. Pro účely globálního zpracování chyb, synchronního nebo asynchronního pre-procesingu požadavku nebo post-procesingu odpovědi, je zapotřebí zachytit požadavky dříve, než jsou předány ke zpracování serveru, a odpovědi dříve, než jsou předány dále aplikačnímu kódu, který inicioval požadavek. K těmto účelům slouží v AngularJS interceptory, které se registrují pomocí **$httpProvideru**, který je přidá do kolekce **$httpProvider.interceptors**. Existují dva druhy interceptorů a dva druhy rejection itenrceptorů (request, requestError, response, responseError). Více o interceptorech je k dispozici na adrese `https://docs.angularjs.org/api/ng/service/$http`. V aplikaci je definovaná "requestHttpInterceptor"továrna, která je přidána do kolekce interceptorů. Pro zpracování chyby ze serveru je použita metoda "responseError", která využije službu "toaster"a zobrazí chybovou hlášku uživateli.

## 5.4 Nasazení

K nasazení systému je nejprve potřeba připravit zázemí. Jedná se o webovou aplikaci, kde serverová část je psána pomocí ASP. NET Web API 2, je tedy potřeba server nebo hosting, který podporuje tuto technologii. Aplikace také potřebuje ukládat data do MS SQL databáze, tu je také potřeba zajistit. Ideální pro takový typ aplikace je mít servery dva a to jeden, kde běží databáze, a druhý aplikační, kde je aplikace nasazena. Protože nevlastním dva různé servery pro nasazení aplikace, byl použit free hosting `http://www.aspone.cz/cz/`.

Nejprve je potřeba vytvořit databázi. Tuto operaci lze provést z klientské části hostingu. Po vytvoření databáze je potřeba vytvořit potřebné tabulky v databázi. Protože používám EntityFramework, pro který existují podpůrné nástroje, lze použít v "Package Manager konzoli"ve vývojovém prostředí Visual Studia powershellový příkaz pro vytvoření databáze. Nicméně hosting nemá povolený vzdálený přístup, proto bylo potřeba powershellovým příkazem "Update-Database -Script"pouze SQL skript vygenerovat, který je nutné manuálně přes webové rozhraní na databázovém serveru pustit.

Po vytvoření databáze stačí získat správný "Connection string", což je řetězec, který slouží pro připojení k databázi, a uložit jej do konfiguračního souboru *Web.Release.config*, který je určený pro aplikaci, která bude nasazena do "produkčního"prostředí. Následně stačí aplikaci vypublikovat pomocí vývojového nástroje, což zajistí překopírování souborů, které jsou výsledkem operace publikování pomocí FTP protokolu na server free hostingu. Aplikace je následně dostupná na adrese `http://travelplanner.aspone.cz/`.

# 6 Výkonnostní a UX testování

Tato kapitola se bude věnovat výkonnostním testům implementovaných biologicky inspirovaných metod v navržené aplikaci a v druhé části se bude zabývat UX experimenty.

## 6.1 Výkonnostní testy

Výkonnostní testy byly provedeny na komponentě mapy. V rámci komponenty jsou implementovány dva biologicky inspirované algoritmy pro optimalizaci tras, jejichž rychlost výpočtu a kvalita byla testována.

Testováno bylo také, jak počet vykreslených bodů a tras ovlivňuje plynulost aplikace v komponentě s mapou a to napříč prohlížeči. Testy byly prováděny na notebooku s procesorem *Intel®Core™ i5-4210M 2.6GHz with Turbo Boost up 3.2GHz*, grafickou kartou *NVIDIA ®GeForce ®840M with 2GB Dedicated VRAM* a 4GB RAM.

Tabulky 5 a 6 obsahují průměrné hodnoty doby trvání výpočtu naměřené při testování obou implementovaných algoritmů pro optimalizaci trasy. Každý algoritmus byl postupně puštěn na sadu "grafů", s rostoucím počtem míst. Oba algoritmy na každém "grafu", byli spuštěny několikrát, aby byli výsledky co nejvíce konzistentní.

Tabulka 5: Výkonnostní testování Ant Colony Optimalization algoritmu

| Počet měst | Rychlost výpočtu | Kvalita výpočtu |
|---|---|---|
| 10 | 5s | 1068,2 km |
| 20 | 21s | 1348,1 km |
| 30 | 32s | 1664,1 km |
| 40 | 91s | 1956,4 km |

Tabulka 6: Výkonnostní testování Inteligent Water Drop algoritmu

| Počet měst | Rychlost výpočtu | Kvalita výpočtu |
|---|---|---|
| 10 | 2s | 1070,9 km |
| 20 | 14s | 1348,1 km |
| 30 | 68s | 1761 km |
| 40 | 104s | 1970,2 km |

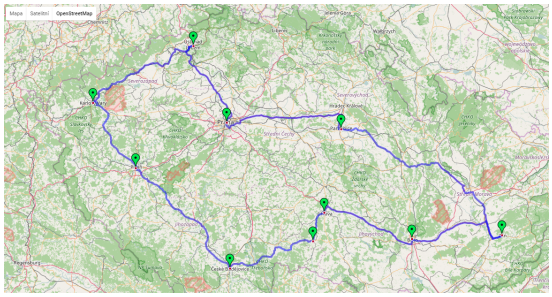
Kvalita nalezené trasy obou algoritmů byla pro "grafy"s počtem míst nepřesahující dvacet určována, pomocí srovnání oproti trase, která byla navrhnuta plánovačem RouteXL popsaného v kapitole 2.1.

Pro "grafy"s větším počtem míst, byla kvalita určena opticky podle nalezené trasy, které neobsahují zvláštní přechody a také naměřené vzdálenosti. Jako referenční trasa byla zvolena

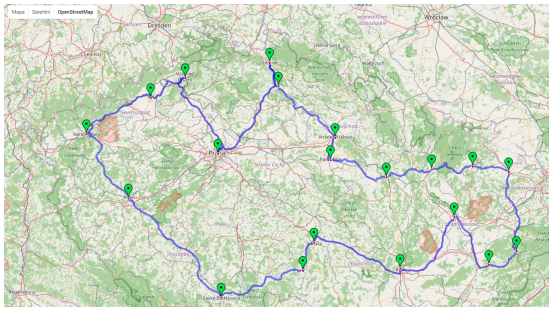
nejkvalitnější z výsledků Ant Colony Optimalization algoritmu, který se ukázal jako stabilnější a také dosahoval lepších výsledků. Kvalita pro Inteligent Water Drop algoritmus byla, určena jako nejlepší nejbližší výsledek, který se nepodařilo několik opakování zlepšit.

Oba algoritmy měli maximální počet iterací nastaven na hodnotu 1000. Pro algoritmus Inteligent Water Drop byl počet iterací zvýšen na hodnotu 1500, pro "grafy"s počtem míst vetší než dvacet, aby algoritmus mohl dosáhnout lepších výsledků. Ant Colony algoritmus používal dvacet mravenčích agentů, kteří hledali řešení a pro Inteligent Water drop algoritmus byl počet IWD roven počtu měst řešeného "grafu".

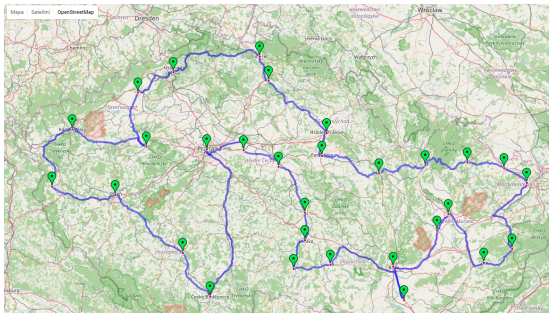
Naměřené hodnoty růstu délky výpočtu v závislosti na počtu míst nejsou nijak překvapivé. S rostoucím počtem míst roste množství kombinací, které musí algoritmy vyhodnotit.



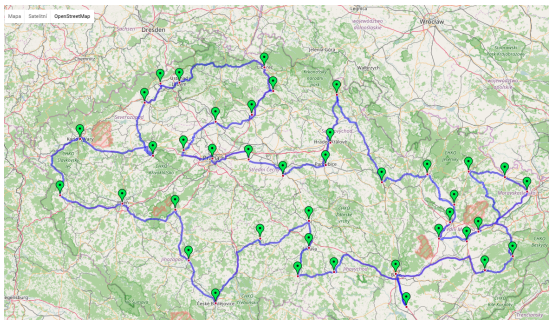
Obrázek 14: Optimální trasa pro 10 míst



Obrázek 15: Optimální trasa pro 20 míst

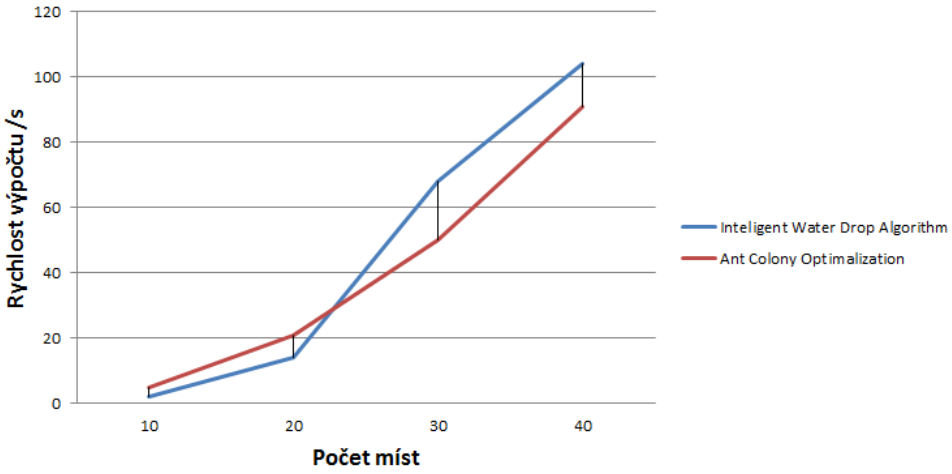


Obrázek 16: Optimální trasa pro 30 míst



Obrázek 17: Optimální trasa pro 40 míst

Optimální trasy pro jednotlivé "grafy"jsou na zobrazeny na obrázcích 14, 15, 16 a 17. Obrázek 18 zobrazuje závislost času výpočtu optimální trasy na použitém algoritmu vzhledem k počtu míst.




Obrázek 18: Doba hledání optimální trasy "grafů"jednotlivými algoritmy

V tabulce 7 jsou uvedeny průměrné hodnoty naměřené při testování vykreslování bodů na mapě. Testování bylo provedeno ve třech prohlížečích a pro různý počet vykreslovaných bodů na mapě.

Tabulka 7: Výkonostní testování vykreslování na mapě

| Prohlížeč | Počet bodů na mapě | FPS |
|---|---|---|
| Google Chrome | 1000 | 17 - 33.8 |
| Google Chrome | 10000 | 9 - 29 |
| Google Chrome | 50000 | 5 - 24 |
| Mozilla Firefox | 1000 | 17 - 34 |
| Mozilla Firefox | 10000 | 15 - 27 |
| Mozilla Firefox | 50000 | 9 - 24 |
| Microsoft Edge | 1000 | 17 - 32 |
| Microsoft Edge | 10000 | 6 - 26 |
| Microsoft Edge | 50000 | 5 - 23 |

Pro každý počet míst byla testována plynulost práce s mapou, kterou jsem měřil jako FPS. Ve všech testovaných prohlížečích lze zapnout ukazatel aktuálního FPS. V testu jsem se zaměřil na běžné úkony s mapou jako je přiblížení, oddálení a posouvaní se na mapě. Pro každý počet míst, bylo určeno rozmezí ve kterém se během práce s mapou pohybovala hodnota FPS.

Výsledky dokazují, že komponenta mapy je dobře optimalizována i pro vetší počet vykreslovaných bodů. Všechny prohlížeče dopadli velice podobně a práce s mapou i pří větším počtu bodů byla bezproblémová. Jediná menší prodleva se ukázala při počtu 50000 míst a přiblížení mapy. Jako horní strop všech testovaných prohlížečů se ukázal počet míst 100000, kdy běžná práce s mapou byla prakticky nemožná.

## 6.2 UX testování

Cílem UX testování vytvořené aplikace je analyzovat chování uživatelů při vykonávání typických úloh v rámci aplikace. Díky uživatelskému testování lze zjistit jakým problémům čelí skuteční uživatelé aplikace. Díky tomu se dozvíme nedostatky uživatelského rozhraní, a můžeme navrhnout konkrétní kroky pro odstranění těchto nedostatků. [17]

Pro testování byla použita metoda "testování použitelnosti". Metoda spočívá ve vytvoření úkolů a k nim příslušných scénářů. Úkol popisuje kroky, které má uživatel provést, ale neříká jak dosáhnout cíle. Scénář popisuje ideální cestu, jak by měl uživatel při plnění úkolu postupovat. Scénáře slouží pro vyhodnocení testování.

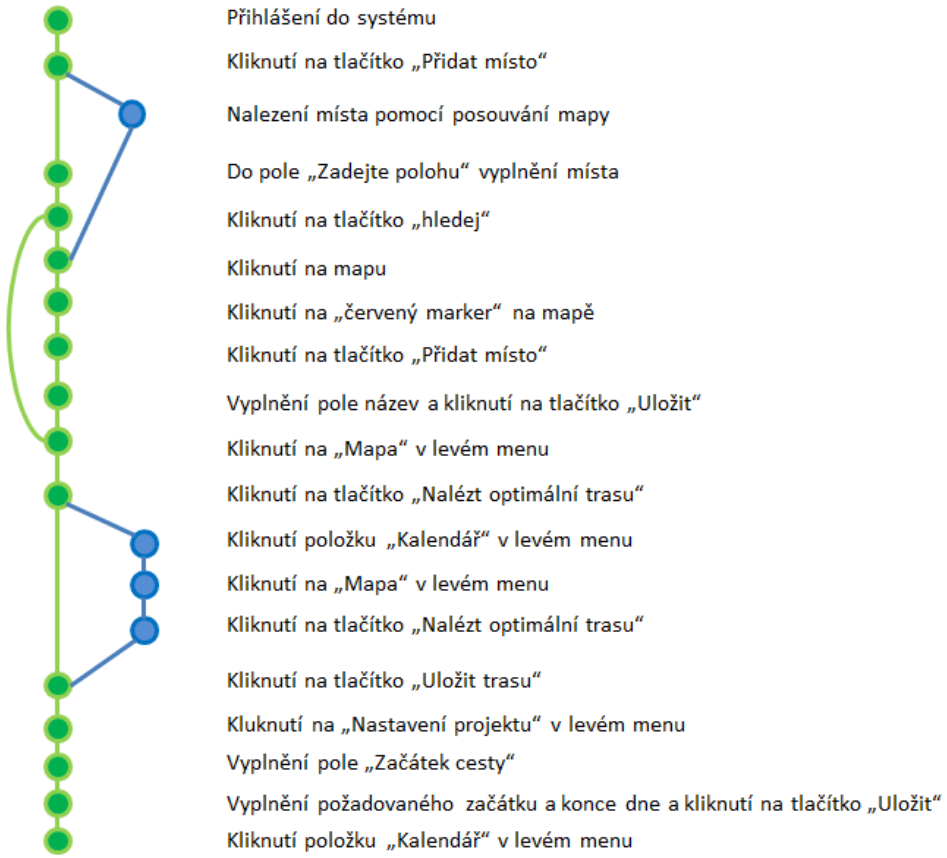
Uživatelé by během testování měli být sami a jejich kroky při práci s aplikací by měli být zaznamenány. Zaznamenávána by neměla být jen obrazovka ale to i jak se uživatel tváří případně jak se pohybují jeho oči po aplikaci. Testování by mělo probíhat ve dvou cyklech s rozestupem několika dní. V prvním testovacím cyklu by uživatel aplikaci neměl vůbec znát abychom zjistili nakolik je aplikace intuitivní. Po dokončení prvního cyklu by měl uživatel být seznámen s aplikací a vysvětlení věcí, kterým nerozuměl. Druhý cyklus je především na otestování zda uživatel porozuměl principům aplikace a zda je pro něj učení jednotlivých principů obtížné. [18]

Po každém cyklu by měli být srovnány činnosti uživatele oproti navrženému scénáři pro daný úkol. Mělo by dojít k vyhodnocení pokud uživatel prováděl některé úkony odlišně, jestli jej naučit správný postup nebo provést změnu v aplikaci a přizpůsobit se práci uživatele.

Nejprve bylo potřeba navrhnout úkoly a vytvořit scénáře. Úkol byl vytvořen jeden postihující základní operace, pro které je aplikace určena:

- **Přidání jednotlivých míst, zobrazení optimální trasy mezi místy a časového harmonogramu** - Vložit informace minimálně o pěti místech a na základě informací o místech získat optimální trasu. Dále zjistit informace o optimální trase. Po uložení optimální trasy, nastavit kolik hodin denně chceme cestovat a zobrazit mezi místy časový harmonogram v kalendáři

Scénář pro úkol je zobrazen na obrázku 19. Díky nedostatečnému technickému zázemí byla sledovaná osoba ve stejné místnosti jako pozorovatel. Problémy nebo nejistota při práci s aplikací byly pouze zapisovány. Poznatky z testování jednotlivých kroků úkolu jsou uvedeny níže a pocházejí z prvního cyklu.




Obrázek 19: Scénář pro definovaný úkol

### 6.2.1 Přidání místa

Na obrázku 19 jsou uvedeny dvě cesty z nichž zelená označuje scénář a druhá odlišnosti, které provedli uživatelé při práci s aplikací. Jako největší problém se ukázalo nalezení kontextového menu při přidání značky na mapu. Uživatelé až na pár ojedinělých případů jej prakticky nebyli schopni bez nápovědy zobrazit.

Negativním zjištěním bylo, že i přestože na úvodní stránce po přihlášení do aplikace je zobrazena "karta"s nápovědou pro přidání místa bylo prvotní přidání místa tak problematický úkon.

Tento problém byl odstraněn okamžitým otevřením kontextového menu jakmile byla značka přidána na mapu. Tato drobná úprava měla pozitivní vliv i na další používání aplikace, protože uživatelé byli schopni zobrazovat detailnější informace, které jsou obsaženy v kontextovém menu po nalezení optimální trasy.

### 6.2.2 Nalezení optimální trasy

Po dokončení přidávání míst přišlo na řadu nalezení optimální trasy. Nalezení tlačítka pro výpočet optimální trasy u některých uživatelů neproběhlo úplně svižně. Nicméně tlačítko bylo ponecháno na stejném místě i po zjištění této skutečnosti, protože jakmile uživatel jednou tlačítko našel, další použití bylo bezproblémové.

Další problém nastal při zobrazení časového harmonogramu trasy. Ukázalo se, že někteří uživatelé díku tomu, že aplikaci dostatečně neznají a nemají představuju o tom jak určité věci fungují vynechávají některé nezbytné kroky. U zobrazení časového harmonogramu často docházelo k tomu, že uživatel po nalezení trasy přeskočil krok uložení optimalizované trasy a rovnou se snažil přejít na stránku s kalendářem. Toto bylo odstraněno informační hláškou, která je uživateli zobrazena po optimalizaci cesty pokud ještě nikdy neprovedl žádné uložení.

Negativním zjištěním opět bylo, že uživatelé opět ignorovali "kartu"s nápovědou pro zobrazení časového harmonogramu na domovské stránce.

Dalším zjištěním bylo, že někteří uživatelé delší čas hledali kde nastavit kolik hodin denně chtějí cestovat. Tento problém byl částečně spojen s neznalostí významu "projekt"a jakou roli v aplikaci hraje. Jakmile bylo uživateli vysvětleno jaký význam "projekt"má bylo toto nalezení nastavení bezproblémové.

Našli se i uživatelé, kteří nejprve proklikali všechny položky v menu a nastavení "projektu"provedli dokonce dříve než začali přidávat nějaká místa.

### 6.2.3 Závěr UX testování

UX testování odhalilo některé problémy. U některých problémů se dá předpokládat, že byly způsobeny úplnou neznalostí systému. Nejvíce se to projevilo při zobrazení harmonogramu, kdy uživatelé netušili, že je nejprve nutné trasu uložit. Dalším problém u aplikace je, že některé zajímavé funkce nejsou na první pohled zřejmé a uživatelé mnohdy na tyto funkce nepřišli. Příkladem může být kalendář, kde po kliku na událost setrvání na místě je zobrazeno dialogové okno s informacemi o místě. Tyto problémy by bylo možné řešit interaktivními tooltipy. Pro nedostatek času tyto tooltipy nebyly implementovány.

Druhý cyklus testování ukázal, že jakmile uživatelé systém znají nemají problém při plnění úkolu nalezení optimální trasy a zobrazení časového harmonogramu.

# 7 Závěr

V rámci této diplomové práce byly představeny dva biologicky inspirované algoritmy, které dokáží řešit problém obchodního cestujícího. Jedním z algoritmů byl Ant Colony Optimalization, a druhým algoritmem byl Inteligent Water Drop Algorithm.

S využitím těchto teoretických znalostí byly, oba algoritmy implementovány. Hlavním výstupem této práce je webová aplikace pro plánování cest a správu jednotlivých míst. Aplikace pro optimalizaci využívá zmíněné biologicky inspirované algoritmy. Práce s aplikací není, nikterak složitá a velmi snadno, lze pro sadu měst nalézt optimální trasu. Aplikace je velmi vázaná na API třetích stran a to zejména na Google Maps API, které je použito pro vykreslování bodů, získávání a vykreslování tras na mapě, ale také pro získávání jednotlivých vzdáleností mezi místy.

Ohledně budoucnosti aplikace a jejího dalšího možného vývoje určitě lze nalézt některé funkce, které by zpříjemnily práci s aplikací, nebo přidali informační hodnotu uživateli. Dále by se určitě dalo zapracovat na optimalizaci obou implementovaných algoritmů, aby jejich časová náročnost nebyla tak vysoká.

Aplikace je nasazena pouze na bezplatném hostingu není to tedy optimální, a prozatím není tedy ani možné uvažovat o rozšíření počtu uživatelů. Získávání vzdáleností, které jsou velmi důležité pro práci obou algoritmů, je také ovlivněno limity, které jsou určeny pro bezplatné použití. Tato funkcionalita by také musela být vyřešena, aby aplikace mohla být rozšířena mezi větší počet uživatelů, minimálně koupí prémiové verze, kterou Google nabízí.

# Literatura


[1] Hamed Shah-Hosseini. The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation*, 1(1-2):71–79, 2009.

[2] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[3] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[4] Christian Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

[5] Xinjie Yu and Mitsuo Gen. Swarm intelligence. *Introduction to Evolutionary Algorithms*, pages 327–354, 2010.

[6] Saad Ghaleb Yaseen and Nada MA Al-Slamy. Ant colony optimization. *IJCSNS*, 8(6):351, 2008.

[7] Sorin C Negulescu, Constantin Oprean, Claudiu V Kifor, and Ilie Carabulea. Elitist ant system for route allocation problem. In *Proceedings of the 8th conference on Applied Informatics and Communications, Greece: World Scientific and Engineering Academy and Society (WSEAS)*, pages 62–67, 2008.

[8] Thomas Stützle and Holger H Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.

[9] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[10] Dervis Karaboga. Artificial bee colony algorithm. *scholarpedia*, 5(3):6915, 2010.

[11] Xin-She Yang. Firefly algorithms for multimodal optimization. In *International Symposium on Stochastic Algorithms*, pages 169–178. Springer, 2009.

[12] Fernando Teixeira Mendes Abrahão and Nicolau Dionísio Fares Gualda. Fleet maintenance scheduling with an ant colony system approach. In *International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 412–419. Springer, 2006.

[13] Sevda DAYIOĞLU GÜLCÜ, Şaban GÜLCÜ, Humar KAHRAMANLI, Alaaddin Keykubat Campus, and Konya Selçuklu. Solution of travelling salesman problem using intelligent water drops algorithm.

[14] Hamed Shah-Hosseini. An approach to continuous optimization by the intelligent water drops algorithm. *Procedia-Social and Behavioral Sciences*, 32:224–229, 2012.

[15] Freeman Adam. *Pro AngularJS*. New York: Apress, 2014. ISBN 9781430264484.

[16] What is entity framework? *Entity Framework Tutorial*. Dostupné z `http://www.entityframeworktutorial.net/what-is-entityframework.aspx`.

[17] Steve Krug. *Don't Make Me Think!* New Riders Publishing Berkeley, California USA, 2006. ISBN 0-321-34475-8.

[18] Rick barron: User experience testing methods. *slideshare*. Dostupné z `https://www.slideshare.net/visionary/user-experience-testing-methods`.